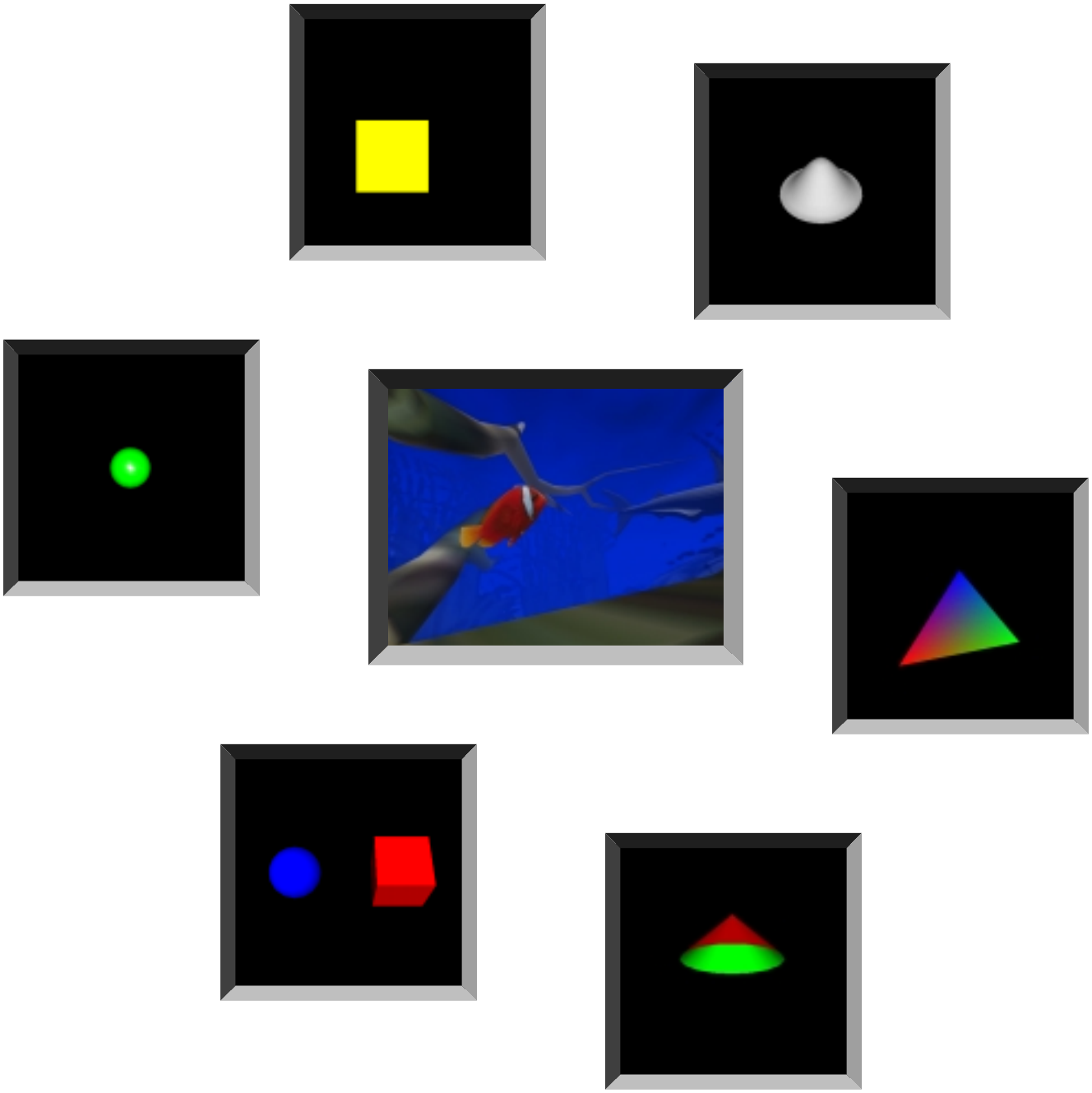


OpenGLプログラミング

NK-EXA Corporation

使用上の注意

本書は、個人、あるいは公的教育機関では、無償でお使いいただけます。ただし、集団利用に付きましては、OpenGL@dst.nk-exa.co.jp まで、ご一報いただけると幸いです。
なお、営利機関における集団教育などでの使用につきましては、
事前に OpenGL@dst.nk-exa.co.jp までご相談下さい。

本文書は上記の限定の下に無償でご利用いただけますが、著作権を放棄するものではありません。本文書の作成には万全を期しますが、明示、暗黙を問わずこの文書の内容に関してはいかなる保証も適用しません。内容に誤りがあり、またこの文書によって、不利益を被っても、一切関知いたしません。また、文書中の登録商標は、一般にその所有者のものです。

その他、本テキスト利用の詳細については、「OpenGL プログラミングコース FAQ」(http://www.nk-exa.co.jp/mmtech/OpenGLEdu/faq.shtml) を御覧ください。

再配布については、ベータ版の間は禁止とさせていただきます。

目 次

第1章　OpenGL概要

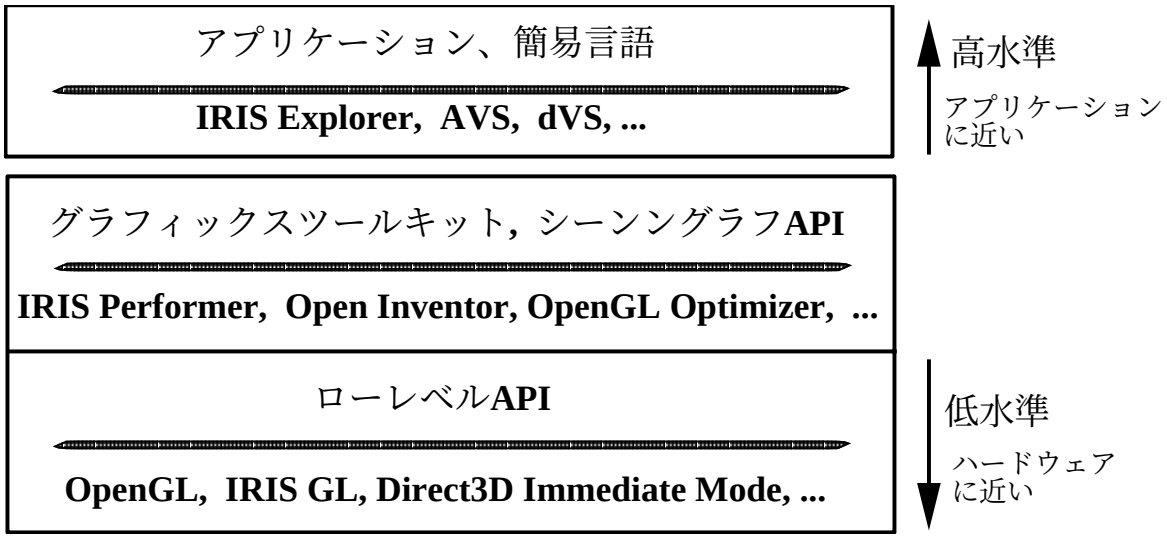
本章の目的

- OpenGL の概要を把握する
- OpenGL プログラミングの手法を理解する
- GLUT ライブラリを使ったウィンドウ操作や、アニメーション処理を理解する
- 簡単な 2 次元プログラムを作成する

1.1　OpenGL 入門

OpenGL とは

- OS、Window システムの壁を越えた唯一の標準 3 次元グラフィックス・ライブラリ
- 名前の由来は Open な Graphics Library
- UNIX(X Window System)、Windows NT/95、OS/2、Mac OS、BeOS で動作
- IRIS GL の技術を継承しているが上位互換性はない
  - OS、ウィンドウシステムから独立
  - イベント処理、ウィンドウ処理はない
- 高い移植性
  - ARB(Architecture Review Board) が仕様管理をし、適合試験を実施している
  - 開発は SGI が担当している
- 強力かつ高品位のプリミティブなライブラリ
  - いわゆるローレベル API であり、高機能なグラフィックス・ツールキットではない
  - グラフィックス・パッケージの階層図を下図に示す



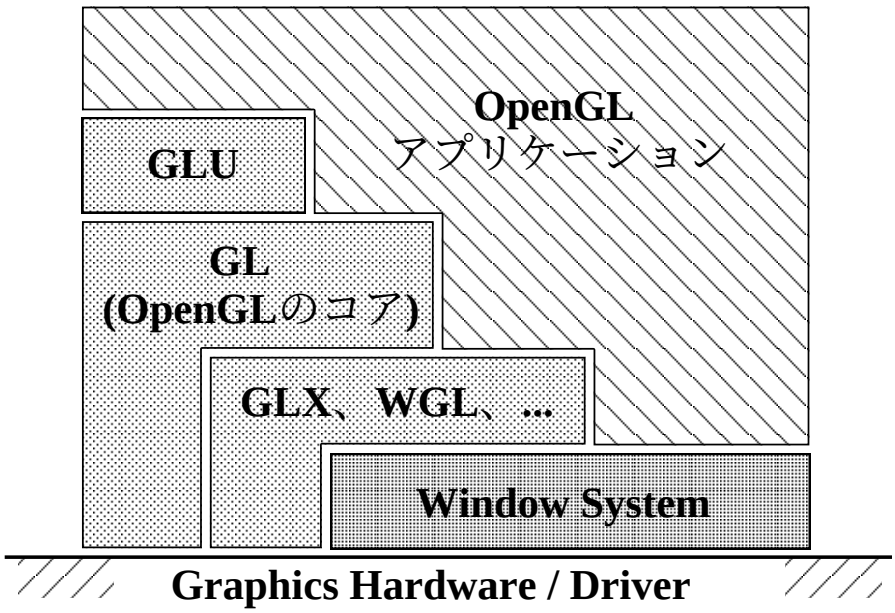
OpenGL の機能

- 基本機能
  - 2 次元、3 次元のジオメトリプリミティブ (点、線、ポリゴン)
  - ラスタプリミティブ (ビットマップ、ピクセルマップ) の操作
  - 各種座標変換と変換マトリックスの操作
  - デプスバッファ・アルゴリズムによる隠面消去
  - Phong ライティングによる光源モデル
  - アルファ合成
  - RGB モードとカラー・インデックス・モード
  - イミディエート・モードとディスプレイ・リスト・モード
  - フィードバック、セレクション、ピッキング

- 高品位レンダリング機能
  - テクスチャ・マッピング
  - 大気効果（霧、霞、スモッグ）
  - モーション・ブラー、焦点深度効果
  - シーン・アンチエリアシング
  - NURBS 曲面
  - 凹状ポリゴン、穴空きポリゴンの 3 角形分解

OpenGL の構成

- OpenGL 製品はコアとなるライブラリ (API) の GL、GL 上のユーティリティ・ライブラリの GLU、ウィンドウ・システムとのインターフェイス・ライブラリの 3 つの主要ライブラリから成る
- GL
  - OpenGL のコアライブラリ
  - OS、Window システムに依存しない
- GLU
  - OpenGL のユーティリティ・ライブラリ
  - NURBS 曲面、複雑なポリゴンの 3 角形分解などの機能を提供
  - OS、Window システムに依存しない
- インターフェイス・ライブラリ
  - OpenGL とウィンドウ・システムのインターフェイス・ライブラリ
  - X Window System では GLX、Windows NT では WGL、OS/2 では PGL、Power Mac では AGL になる
  - OS、Window システムに依存する
- OpenGL の各ライブラリの階層構造は次の通り



OpenGLの関連ツール

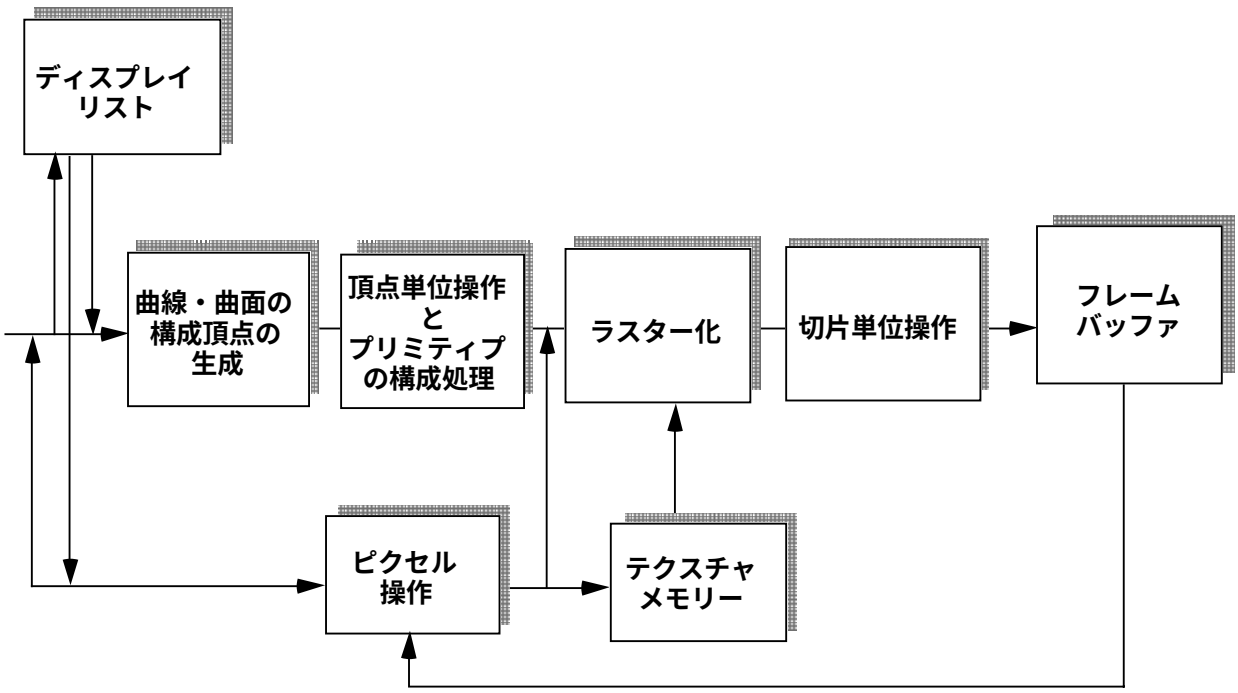
- GLUT
  - OpenGL 初期化、ウィンドウ操作、イベント処理を行うライブラリ
  - Fortran バインディングもサポートされている
  - OpenGL の Fortran 版 API は IRIX5.3 ではサポート
  - このテキストではウィンドウ操作、イベント処理に GLUT ライブラリを使用する
  - 次の AUX に対し機能的に上位互換のライブラリであるが、本格的な GUI を使ったアプリケーションには向かない
- AUX
  - すべての OpenGL 製品に含まれる (X、Windows NT、OS/2...)
  - GLUT に置き換わられている
  - OpenGL 初期化、ウィンドウ操作、イベント処理を行う教育用のライブラリ
  - AUX ライブラリには TK ライブラリというよりプリミティブなライブラリが含まれる
    （Tcl/Tk の Tk とは全く無関係であるので注意）
- MESA
  - OpenGL 互換を目指すフリーのライブラリ
  - 100%互換ではない
  - Linux、FreeBSD といった PC 上の UNIX から、IRIX、Solaris、AIX、HP/UX などほとんどの UNIX 上で動作
  - ftp://iris.ssec.wisc.edu/pub/Mesa
- FLTK
  - C++用の GUI 作成ツールキット
  - ディジタルド メイン社内用 GUI 作成ツールをフリーウェアとして公開
    http://www.cinenet.net/users/spitzak/fltk/
  - X Window System と、Windows の両者をサポート
  - X 専用の XForms というものもある
    http://bragg.phys.uwm.edu/xforms

OpenGL ステートマシン

- OpenGL は仮想的なステート・マシンとして実装されている

- OpenGL ステート・マシンが状態を遷移しながら描画を進めていく
  - 状態設定関数の呼び出しと描画関数の呼び出しを繰り返す
  - 関数の引数を少なくし効率的なプログラミングができる
  - 状態設定が違うと同じ描画関数でも異なる結果になる
  - OpenGL は簡潔に表現できるがコンテキスト (文脈) に依存する
    つまり日本語的

- OpenGL の構造



OpenGLプログラミングをはじめる前に

- 本教育コースはOpenGL のプログラミング学習を通して3 次元グラフィックスの基礎を習得することを目的にしている
  - 3 次元グラフィックス・プログラミングには下記に示すように、OpenGL 上のグラフィックス・ツールを使う方法などもあるが、どの場合にも本コース程度の OpenGL ならびに3 次元グラフィックスの基礎知識は必要となる
- OpenGL を直接使うプログラミングは次のような場合に向いている
  - 独自のグラフィックス・ツールキットやグラフィックス・ライブラリの作成
  - 独自のシーン・データベース形式が必要なプログラム
  - ポータビリティーや、低コスト、あるいは徹底したチューニングが要求されるプログラム
  - 3 次元グラフィックスの基礎学習や小物プログラム
- OpenGL 上のグラフィックス・ツールは次のものが用意されている

OpenGL Optimizer

- Digital Prototyping、Virtual Mock-up 用のオブジェクト指向 API(C++)
- CAD、Scientific Visualization などの大規模データを対話処理可能な速度で可視化する

OpenGL Volumizer

- ボリュームレンダリング用のオブジェクト指向 API
- MRI、CT スキャナーなどの連続断層写真から3 次元オブジェクトを再構成

IRIS Performer

- ビジュアル・シミュレーション専用の3 次元グラフィックス・パッケージで IRIX 版と Linux 版が用意されている (C++/C)
- Culling、LOD、マルチプロセッサ処理、フレームレート維持などをサポート
- 各種シーン・データベースの読み込みをサポート（保存はできない）

Open Inventor

- オブジェクト指向の汎用の高機能3 次元グラフィックス・パッケージで、マルチ・プラットフォーム化されている (C++/C)
- 階層型のシーン・データベースを中心に3 次元物体操作、高精度ピッキング、印刷から Culling、LOD(Level Of Detail) までサポート

ImageVision Library

- イメージ・プロセッシング（画像処理）専用のオブジェクト指向グラフィックス・パッケージで、C++、C、Fortran の API が用意されており、C++によりライブラリの拡張が可能
- SGI 専用だが、マルチ・プラットフォーム化が検討されている

1.2　OpenGLプログラミング初歩

簡単なOpenGLプログラムの流れ図

- GLUT ライブラリを使った簡単な OpenGL プログラムの流れ図を示す

ヘッダー・ファイルとライブラリの指定

- 個別関数の説明の前にヘッダーファイルとライブラリの指定を説明する
- ヘッダーファイル
  OpenGL のライブラリ関数を使う場合は次の 3 種類のヘッダー・ファイルを必要に応じてインクルードする

```
#include <GL/gl.h>        /* 必ず入れる */
#include <GL/glu.h>       /* glu ライブラリを使うとき */
#include <GL/glx.h>       /* X Window System の場合 */
```

- GLUT ライブラリを使う場合は次のファイルだけをインクルードする

```
#include <GL/glut.h>
```

  – gl.h や glu.h は glut.h の中でインクルードされる
  – glut.h が、環境に応じて windows.h を適切な順番でインクルードするので、UNIX と Windows でポータブルなソースコードになる
- IRIX ではライブラリのリンクには次のオプションが必要である

```
-lglut -lGLU -lGL -lX11
```

  – GLUT ライブラリの関数は libglut.a に、GLU は libGLU.so、GL と GLX は libGL.so に含まれる
  – libX11.so は x11 の関数が含まれ OpenGL プログラムのリンクには必要

  例 `simple.c` というプログラムから `simple` という実行モジュールを作成する

```
% cc simple.c -o simple -lglut -lGLU -lGL -lX11
```

- OpenGL の関数名、定数名、タイプ名に関しては規約があり、規則的に名前が付けれている
  (詳しくは付録 D を参照のこと)

画面のクリア

- 画面をクリアする色の指定

  ```
  glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);
  ```

  – GLclumpf は 0.0f から 1.0f の間の単精度浮動小数点
  – alpha は透明度を示すパラメータ

  例 画面をクリアする色を黒に設定 `glClearColor(0.0f, 0.0f, 0.0f, 0.0f);`

- 画面を glClearColor で指定した色でクリアする

  ```
  glClear(GL_COLOR_BUFFER_BIT);
  ```

- バッファリングされている描画コマンドをすべてサーバに送る

  ```
  glFlush();
  ```

  – 1 つのシーンに対する描画処理が終った時点でコールする
  – ダブルバッファー・モードでは不要

ウィンドウの初期設定(1)

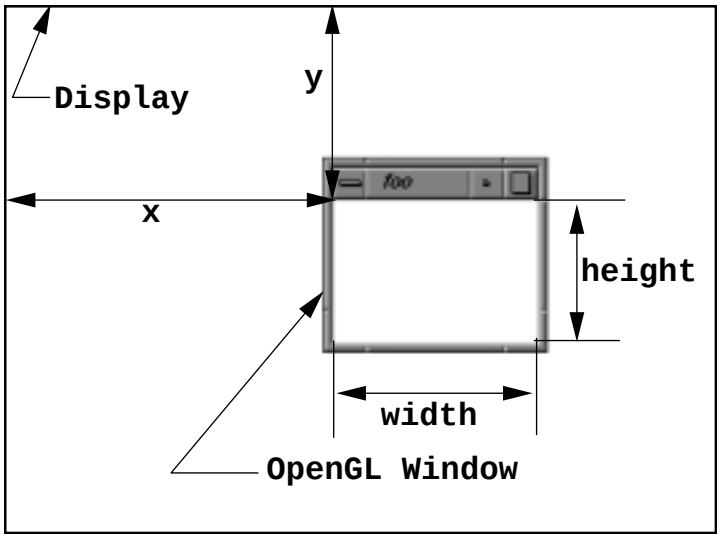
- 本コースではウィンドウ操作、イベント処理に GLUT ライブラリを使う
  - ラスターフォントとベクトルフォントを使った文字列出力機能をサポート
  - 球、円筒、多面体などの基本的な 3 次元図形の描画をサポート
    (GLUT ライブラリの基本図形はディスプレイリスト化もできる)
  - GLUT ライブラリの関数は頭に glut というプレフィックスが付く
- GLUT の初期化を glutInit で行う

```
void glutInit(int *argcp, char **argv);
```

  - argcp には main 関数からコマンドライン引数を示す argc のアドレスを渡す
    GLUT で解釈した引数については、その数を減じる
  - argv は main 関数の argv そのものである
    GLUT で解釈した引数は、配列から消去される
  - X Window System 上への実装の場合は、GLUT の解釈するコマンドライン引数は-display, -geometry などがある (詳細は man X)
- glutInitWindowPosition でウィンドウを開く位置の指定

```
void glutInitWindowPosition(int x, int y);
void glutInitWindowSize(int width, int height);
```

  x、y はウィンドウ内側の左上の点のスクリーン座標値
  width と height はスクリーン上のピクセル数でウィンドウの幅と高さを指定



- glutInitWindow*関数がウインドウシステムと連絡して OpenGL 初期化処理をする
  glutInitWindow*関数のパラメータで指定する座標系はウインドウシステム固有のものを使い、画面左上を原点とする (OpenGL の座標系の取り方とは異なる)

ウィンドウの初期設定 (2)

- ウィンドウの表示モードの初期化を glutInitDisplayMode で行う

  `void glutInitDisplayMode(unsigned int mode);`
  `mode` は以下に示すフラグ

  – RGB モードかカラー・インデックス・モードか（GLUT_RGB, GLUT_INDEX）
  – アニメーションをするかどうか（GLUT_SINGLE, GLUT_DOUBLE）
  – 隠面消去をするかどうか（GLUT_DEPTH）
  – ステンシル・プレーンを使うかどうか（GLUT_STENCIL）
  – 必要なフラグを or（|）で繋いで指定
  　例えば、RGB モードで隠面消去してアニメーションをするときは、

  ```
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
  ```

  – デフォルトはカラー・インデックスでシングル・バッファ
  　（GLUT_INDEX |GLUT_SINGLE）

- OpenGL ウィンドウを初期化して開く
  OpenGL を使って描画するためのキャンバスとなるウィンドウを OpenGL ウインドウと呼ぶ

  `int glutCreateWindow(char *titleString);`
  ウィンドウのタイトルを `titleString` で指定する

- イベント処理の無限ループに入る

  `void glutMainLoop(void);`

  – イベント処理の無限ループに入る
  – glutDisplayFunc などの各種コールバック関数でイベントに対する処理を指定する

イベント処理関数の登録

- glutMainLoop の無限ループ内で発生したイベントを処理する関数を、ループに入る前に予め登録する
  - イベント処理関数は、コールバック関数とも呼ばれる

- 描画イベントが発生したときに処理を行う関数を登録する

```
void glutDisplayFunc(void (*func)(void));
```

  - func は、引数も戻り値もない関数へのポインタ
  - ここで登録した処理関数で、OpenGL の描画関数を呼び出す

- キーボード イベントが発生したときに処理を行う関数を登録する

```
void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));
```

  - funk は、key、x、y と 3 つの引数を持ち、戻り値のない関数へのポインタ
  - key にはキーボード イベントを発生させたキーのコードが入る
  - x、y にはキーボード イベントが発生したときのマウスカーソルのウインドウ座標が入る

- ウィンドウ・サイズに変更があったときの処理を行う関数の指定

```
void glutReshapeFunc(void (*func)(int width, int height));
```

  - funk は、width と height の 2 つの引数を持ち、戻り値のない関数へのポインタ
  - width と height にはそれぞれ、OpenGL ウィンドウの幅と高さのピクセル数が渡される

```
 1 /*
 2 ** clr.c - 画面を黒でクリアする
 3 */
 4
 5 #include <GL/glut.h> /* GLUT ライブラリ用 */
 6
 7 /* 初期化 */
 8 void
 9 initialize(void)
10 {
11   /* 画面クリアカラーの設定 */
12   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
13 }
14
15 /* 描画 */
16 void
17 display(void)
18 {
19   /* 画面クリア */
20   glClear( GL_COLOR_BUFFER_BIT );
21
22   /* 描画終了 */
23   glFlush();
24 }
25
26 /* キーボード入力処理 */
27 void
28 keyboard( unsigned char c, int x, int y )
29 {
30   /* ESC キー (コード 27) でプログラム終了 */
31   if( c==27 ) exit( 0 );
32 }
33
34 /* ウィンドウ変形処理 */
35 void
36 reshape( int w, int h )
37 {
38   /* 何もしない */
39 }
40
41 int
42 main( int argc, char *argv[] )
43 {
```

```
44   /* glut のイニシャライズ */
45   glutInit( &argc, argv );
46
47   /* ウィンドウの生成 */
48   glutInitDisplayMode( GLUT_RGB );
49   glutInitWindowPosition( 100, 100 );
50   glutInitWindowSize( 500, 400 );
51   glutCreateWindow( argv[0] );
52
53   /* 描画ルーチンの設定 */
54   glutDisplayFunc( display );
55
56   /* キーボード入力処理ルーチンの設定 */
57   glutKeyboardFunc( keyboard );
58
59   /* ウィンドウ変形処理ルーチンの設定 */
60   glutReshapeFunc( reshape );
61
62   /* OpenGL 初期化ルーチンの呼び出し */
63   initialize();
64
65   /* 無限ループ */
66   glutMainLoop();
67   return 0;
68 }
69
```

<演習>

1. forYou ディレクトリの、clr プログラムを make、実行して下さい。

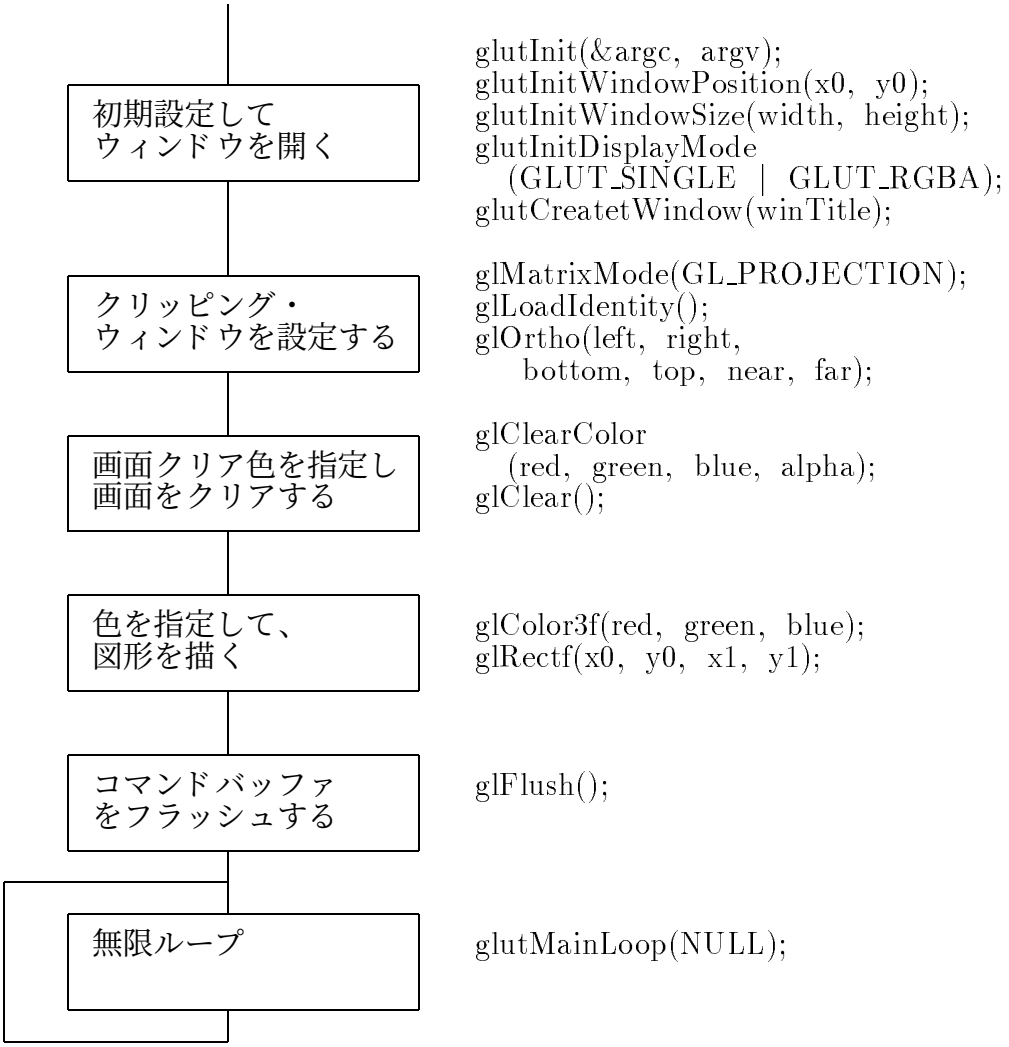
```
% cd forYou
% make clr
% clr
```

2. 作業ディレクトリを作成し、そこに clr プログラムをコピーして、ウィンドウサイズやクリア色を変更して下さい。

```
% cd ~
% mkdir bobby
% cp forYou/clr.c forYou/Makefile bobby
% cd bobby
% vi clr.c
...（適当にプログラムの変更を行なう）
% make clr
% clr
```

1.3 2 次元図形の描画とクリッピング・ウィンドウ

長方形の描画をするプログラムの流れ図



描画関数

- 色の設定

```
glColor3f(GLclampf red, GLclampf green, GLclampf blue);
```

  – 0.0f から 1.0f の範囲の 3 つの浮動小数点数で赤、緑、青の強さを指定
  例 黄色を指定
  `glColor3f(1.0f, 1.0f, 0.0f);`

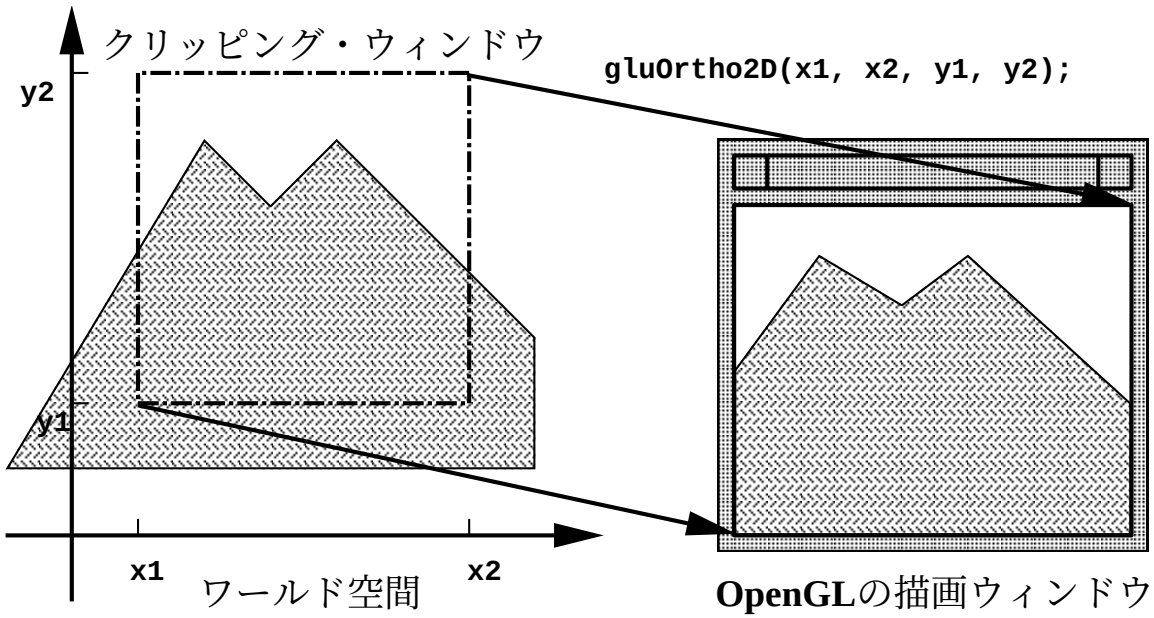
- 長方形の描画

```
glRectf(x0, y0, x1, y1);
```

  – 左下の点 (x0, y0) から、右上の点 (x1, y1) の塗り潰しの長方形を描く

  例 (-0.5, -0.5)、(0.5, 0.5) の 2 点を対角にした長方形を描く
  `glRectf(-0.5, -0.5, 0.5, 0.5);`

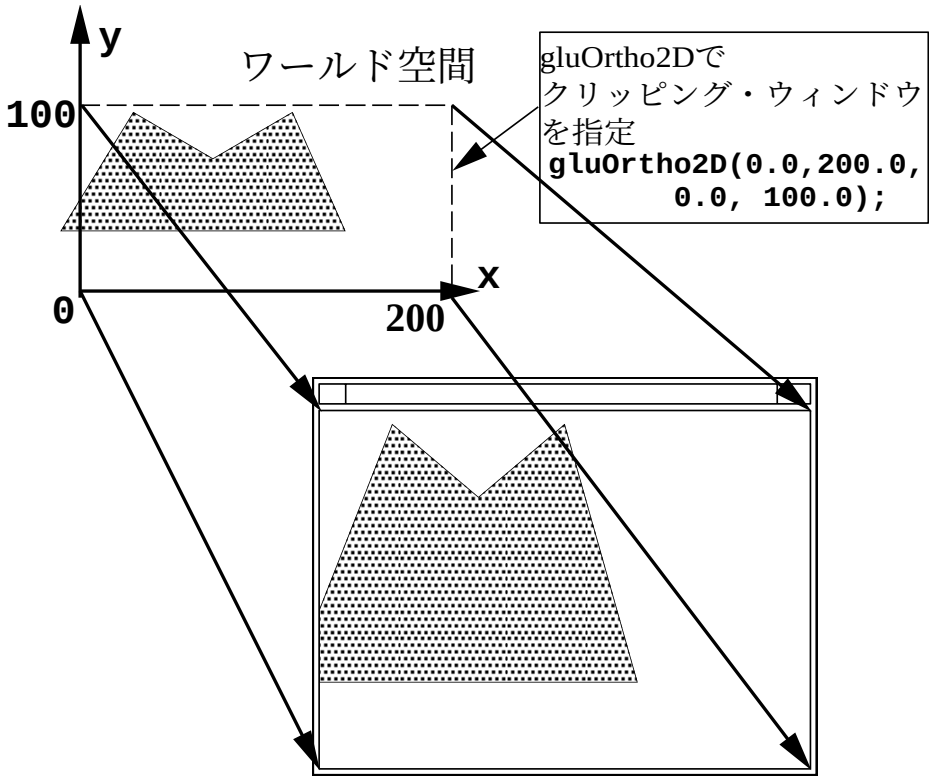
クリッピング・ウィンドウと図形の描画



- ユーザーはワールド空間という仮想的な空間の任意の位置に図形を描くことができる
  - ワールド空間は第 1 象限を右上部とする右手座標系
- はじめにワールド空間中の図形を描こうと思う領域に、クリッピング・ウィンドウを設定する
  - クリッピング・ウィンドウ内部に描いた図形だけが、ディスプレイ上の OpenGL の描画ウィンドウにマッピングされる
  - クリッピング・ウィンドウの設定は gluOrtho2D 関数を使う

クリッピング・ウィンドウのマッピング

- クリッピング・ウィンドウと OpenGL の描画ウインドウの縦横比が異なると図形は歪んでウインドウにマッピングされる
- 図形を歪ませないようにするコントロールはアプリケーションが行う



OpenGLの描画ウィンドウ

クリッピング・ウィンドウ設定の前準備

- OpenGL には 3 つの変換マトリックスがあり、それぞれ投影変換、モデルビュー変換、テクスチャー座標変換に使う
  マトリックスの設定は 3 つの中からカレントの処理対象マトリックスとして指定されているもの 1 つに対して行う
  この設定をマトリックスモードの設定という
- クリッピング・ウィンドウを設定するために、予めマトリックスモードを投影変換マトリックスにして処理を行う
- マトリックスモードの設定関数

```
void glMatrixMode(GLenum mode);
```

  – glMatrixMode() の mode 引数で操作対象のマトリックスを指定する
    GL_PROJECTION 投影変換マトリックスを操作対象とする
    GL_MODELVIEW モデルビュー変換マトリックスを操作対象とする
    GL_TEXTURE テクスチャー座標変換マトリックスを操作対象とする
- マトリックスモードの変換のマトリックスを単位マトリックスで初期化する関数

```
void glLoadIdentity(void);
```

- 次の手順でクリッピング・ウィンドウを設定する
  1. マトリックスモードを投影変換のマトリックスにする
     `glMatrixMode(GL_PROJECTION)`
  2. 投影変換のマトリックスを単位マトリックスで初期化
     `glLoadIdentity(void);`
  3. クリッピング・ウインドウの設定を行う（次ページで説明）
  4. 投影変換の処理が終ったら処理対象マトリックスをモデルビュー・マトリックスにしておく
     `glMatrixMode(GL_MODELVIEW);`

クリッピング・ウィンドウの設定関数

- 2 次元ワールド空間中にクリッピング・ウィンドウ（描画エリア）を設定する

```
void gluOrtho2D(GLdouble left, GLdouble right,
                GLdouble bottom, GLdouble top);
```

  - クリッピング・ウィンドウ内に描いたものだけがディスプレイのウィンドウ内に表示される
  - gluOrtho2D 関数は 3 次元平行投影変換を設定する glOrtho 関数の引数 near、far をそれぞれ−1.0 と 1.0 に設定した場合と同じである
    すなわち OpenGL の 2 次元描画は 3 次元の特殊な場合として処理されている
    例 各方向-1.0 から 1.0 からの範囲に描画するとき

```
gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
```

    または

```
glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
```

- まとめると 2 次元描画に必要なクリッピング・ウィンドウの設定は次の通り

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
glMatrixMode(GL_MODELVIEW);
```

```
 1 /*
 2 ** draw.c - 黄色い四角形を描画する
 3 */
 4
 5 #include <GL/glut.h>
 6
 7 void
 8 initialize(void)
 9 {
10   /* クリッピングウィンドウの設定 */
11   glMatrixMode( GL_PROJECTION );
12   glLoadIdentity();
13   gluOrtho2D( -100.0, 100.0, -100.0, 100.0 );
14   glMatrixMode( GL_MODELVIEW );
15
16   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
17 }
18
19 void
20 display(void)
21 {
22   glClear( GL_COLOR_BUFFER_BIT );
23
24   /* 黄色い四角形の描画 */
25   glColor3f( 1.0f, 1.0f, 0.0f );
26   glRectf( -80.0f, -80.0f, -20.0f, -20.0f );
27
28   glFlush();
29 }
30
31 void
32 keyboard( unsigned char c, int x, int y )
33 {
34   if( c==27 ) exit( 0 );
35 }
36
37 void
38 reshape( int w, int h )
39 {
40 }
41
42 int
43 main( int argc, char *argv[] )
```

```
44 {
45   glutInit( &argc, argv );
46
47   glutInitDisplayMode( GLUT_RGB );
48   glutInitWindowPosition( 100, 100 );
49   glutInitWindowSize( 500, 400 );
50   glutCreateWindow( argv[0] );
51
52   glutDisplayFunc( display );
53   glutKeyboardFunc( keyboard );
54   glutReshapeFunc( reshape );
55
56   initialize();
57   glutMainLoop();
58   return 0;
59 }
60
```

<演習>

1. forYou ディレクトリの draw プログラムを実行して下さい。

```
% cd forYou
% make draw
% draw
```

2. 作業ディレクトリに draw プログラムをコピーして、gluOrtho2D() のパラメーターを適当に変更し、見え方の変化を確認して下さい。

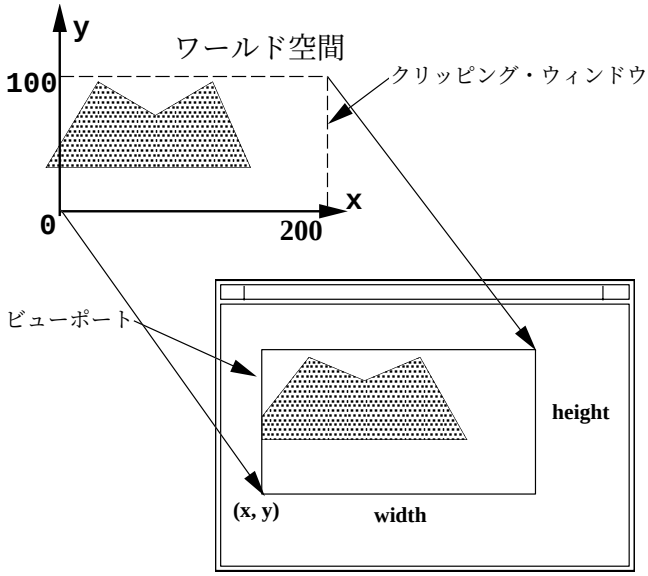
```
% cd ~
% cp forYou/draw.c bobby
% cd bobby
% vi draw.c
...（適当にプログラムの変更を行なう）
% make draw
% draw
```

3. さらに draw プログラムの実行中にウィンドウ形状を変化させて、絵がどのようになるか観察して下さい。

ウィンドウ・サイズの変更とビューポートの設定

- OpenGL の描画ウィンドウの中で、シーンのマッピングされる矩形領域をビューポートと呼ぶ
  - OpenGL の描画ウィンドウが最初に開いた時点ではビューポートは描画ウィンドウに一致している
  - 描画ウインドウのサイズが変更になった場合は、それに応じてビューポートの再設定が必要になる（通常は描画ウィンドウに一致させる）
- ビューポート設定関数
  void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);

  - (x, y) は左下端のディスプレイ座標値、width と height はそれぞれピクセル単位でのビューポートの幅と高さ
  - ウィンドウサイズに変更があったときに呼ばれる、イベント処理関数 (コールバック関数) の中で、glViewport を使ってビューポートの再設定を行う
    例えば GLUT を使う場合は、glutReshapeFunc で指定した関数



OpenGLの描画ウィンドウ

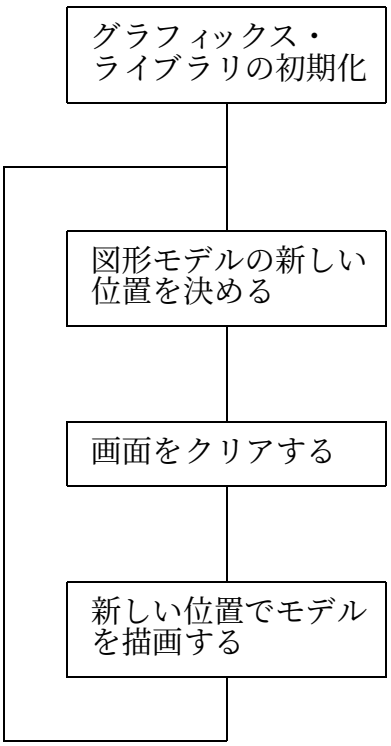
```
 1 /*
 2 ** reshape.c - ウィンドウ変形処理を行なう
 3 */
 4
 5 #include <GL/glut.h>
 6
 7
 8 void
 9 initialize(void)
10 {
11   /* クリッピングウィンドウの初期設定は、
12      最初にreshape()が呼ばれる時に行なわれる */
13   /*
14     glMatrixMode( GL_PROJECTION );
15     glLoadIdentity();
16     gluOrtho2D( -100.0, 100.0, -100.0, 100.0 );
17     glMatrixMode( GL_MODELVIEW );
18   */
19
20   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
21 }
22
23 void
24 display(void)
25 {
26   glClear( GL_COLOR_BUFFER_BIT );
27
28   glColor3f( 1.0f, 1.0f, 0.0f );
29   glRectf( -80.0f, -80.0f, -20.0f, -20.0f );
30
31   glFlush();
32 }
33
34 void
35 keyboard( unsigned char c, int x, int y )
36 {
37   if( c==27 ) exit( 0 );
38 }
39
40 void
41 reshape( int w, int h )
42 {
43   /* ビューポートの再設定 */
```

```
44   glViewport( 0, 0, w, h );
45
46   /* マトリックスの再構築 */
47   glMatrixMode( GL_PROJECTION );
48   glLoadIdentity();
49   gluOrtho2D( -100.0, 100.0, -100.0, 100.0 );
50   glMatrixMode( GL_MODELVIEW );
51 }
52
53 int
54 main( int argc, char *argv[] )
55 {
56   glutInit( &argc, argv );
57
58   glutInitDisplayMode( GLUT_RGB );
59   glutInitWindowPosition( 100, 100 );
60   glutInitWindowSize( 500, 400 );
61   glutCreateWindow( argv[0] );
62
63   glutDisplayFunc( display );
64   glutKeyboardFunc( keyboard );
65   glutReshapeFunc( reshape );
66
67   initialize();
68   glutMainLoop();
69   return 0;
70 }
71
```

1.4 GLUT ライブラリを使ったアニメーション

アニメーション・プログラムの処理フロー

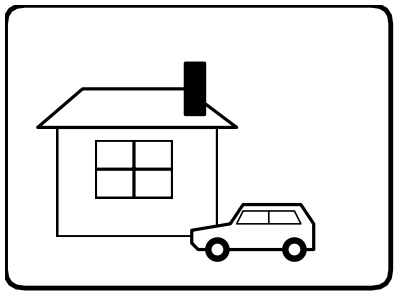
- アニメーションのプログラムはループを構成し、その中で図形の位置や形状を変更して描画する




```
...
glutInitDisplayMode
      (GLUT_SINGLE | GLUT_RGBA);
...
```

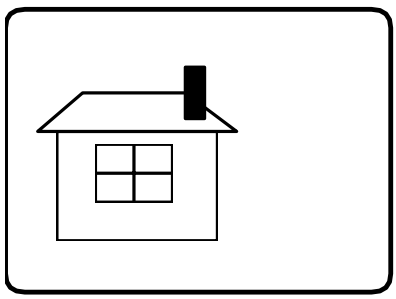
シングル・バッファリングと ダブル・バッファリング

- 下図に示すシングル・バッファーでは画面書換えの処理過程が見えてしまいちらついてスムーズなアニメーションにならない



画面のクリア

家を描く

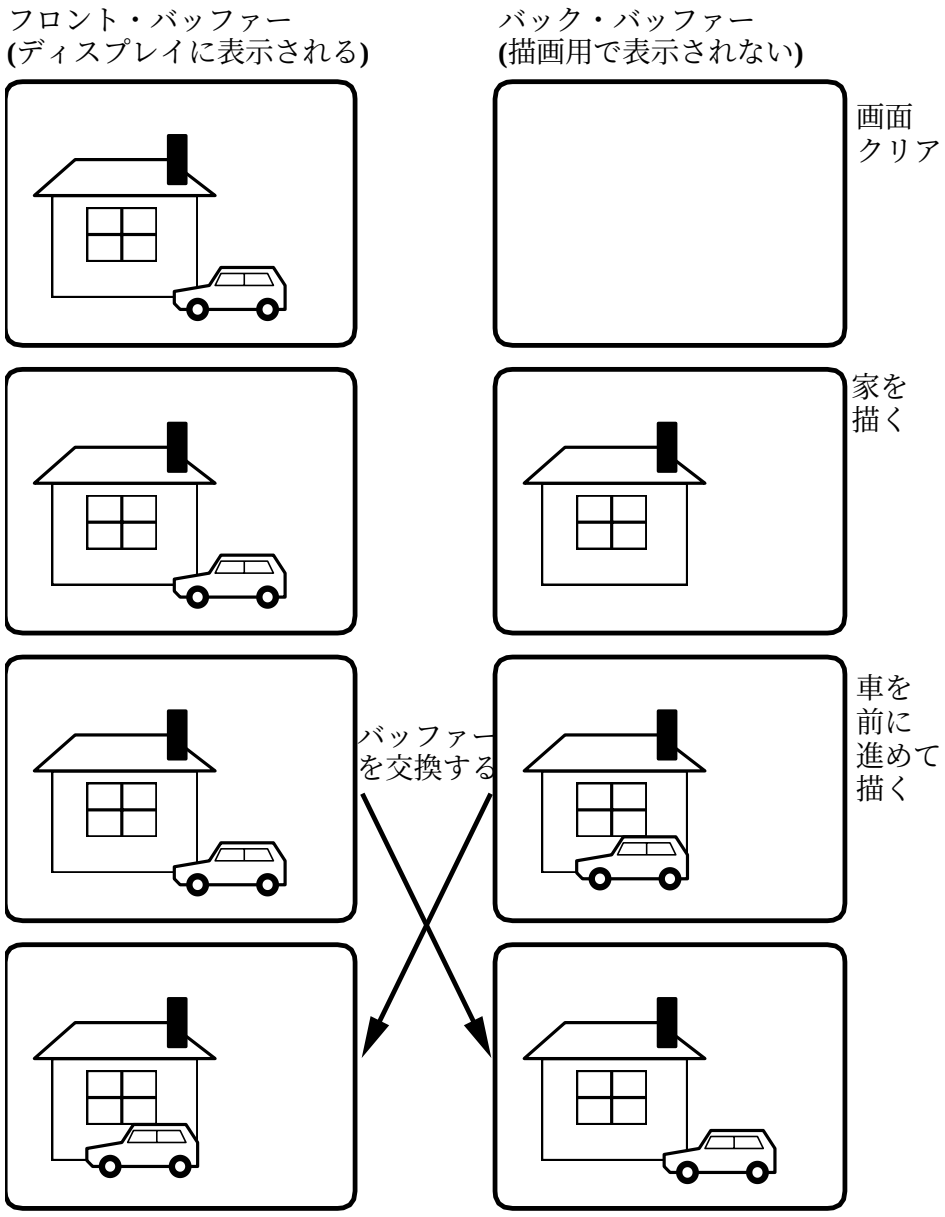


車を前に進めて描く

- ちらつきを除去するためアニメーション・プログラムにはダブル・バッファーモードを使用する

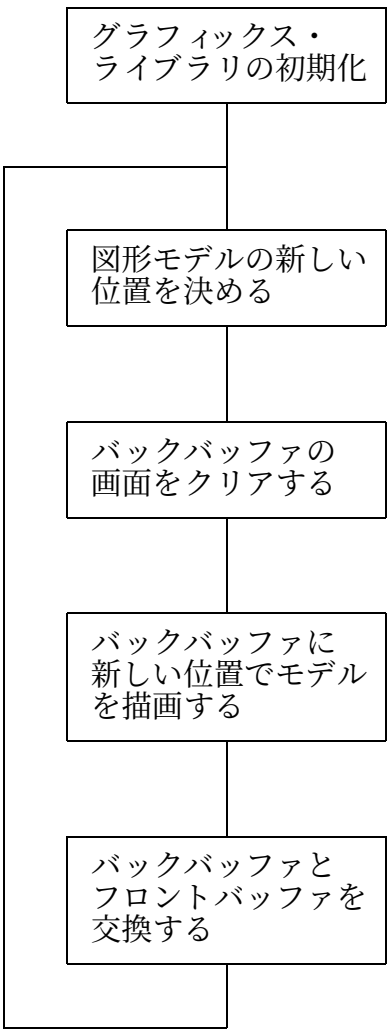
ダブルバッファリングによるアニメーション

- ディスプレイの表示はフロント・バッファーを使う
  一方、描画はバックバッファーに対して行う



ダブルバッファリングのプログラミングの流れ

- 前ページの図を流れ図にすると次のようになる




```
...
glutInitDisplayMode
      (GLUT_DOUBLE | GLUT_RGBA);
glutIdleFunc(...);
```

```
glutSwapBuffers();
```

ダブルバッファ・モードの設定

- ウィンドウ初期化時にダブルバッファ・モードを設定
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  - IRIS GL と異なり OpenGL では 1 つのウィンドウを動的にシングルバッファからダブルバッファに切替えることはできない
    （2 つのウィンドウで切替えることで対応する）
- glutSwapBuffers はダブルバッファー・モードのときフロント・バッファーとバック・バッファーの入れ替えを行う

```
void glutSwapBuffers(void);
```

- glutIdleFunc はイベントが何も発生していない場合に、ループの最後で処理を行う関数を登録する

```
void glutIdleFunc(void (*func)());
```

  - func は、引数も戻り値もない関数へのポインタ
  - アニメーション表示のとき、func 関数から次の glutPostRedisplay を呼んで描画イベントを強制発生させるのに使う
  - アニメーション表示において位置の再設定などに使う

- 描画イベントを発生させる (ポストする)

```
void glutPostRedisplay(void);
```

  - カレントウィンドウのノーマルプレーンの再描画が必要であるというイベントを発生させる
  - 次のイベント処理ループで描画イベント処理関数 (glutDisplayFunc で登録されたもの) が呼ばれる
    (glutDisplayFunc で指定した関数)
  - 1 つのイベント処理ループの中で描画イベントは 1 つにまとめられる

```
 1 /*
 2 ** single.c - シングルバッファでアニメーションを行なう
 3 */
 4
 5 #include <GL/glut.h>
 6
 7 void
 8 initialize(void)
 9 {
10   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
11 }
12
13 void
14 display(void)
15 {
16   /* アニメーション用変数 */
17   static float offsetX = -1.0;
18
19   /* アニメーション用変数の計算 (offsetX: 0.0~99.0) */
20   offsetX += 1.0;
21   if( offsetX>=100.0 ) offsetX -= 100.0;
22
23   glClear( GL_COLOR_BUFFER_BIT );
24
25   glColor3f( 1.0f, 1.0f, 0.0f );
26
27   /* offsetX 分ずれた長方形を描画 */
28   glRectf( -80.0f+offsetX, -80.0f, -20.0f+offsetX, -20.0f );
29
30   glFlush();
31 }
32
33 void
34 keyboard( unsigned char c, int x, int y )
35 {
36   if( c==27 ) exit( 0 );
37 }
38
39 void
40 reshape( int w, int h )
41 {
42   glViewport( 0, 0, w, h );
43
```

```
44   glMatrixMode( GL_PROJECTION );
45   glLoadIdentity();
46   gluOrtho2D( -100.0, 100.0, -100.0, 100.0 );
47   glMatrixMode( GL_MODELVIEW );
48 }
49
50 /* アニメーション描画処理 */
51 void
52 animate(void)
53 {
54   glutPostRedisplay();
55 }
56
57 int
58 main( int argc, char *argv[] )
59 {
60   glutInit( &argc, argv );
61
62   glutInitDisplayMode( GLUT_RGB );
63   glutInitWindowPosition( 100, 100 );
64   glutInitWindowSize( 500, 400 );
65   glutCreateWindow( argv[0] );
66
67   glutDisplayFunc( display );
68   glutKeyboardFunc( keyboard );
69   glutReshapeFunc( reshape );
70
71   /* アニメーション描画処理ルーチンを
72      アイドリングルーチンとして設定 */
73   glutIdleFunc( animate );
74
75   initialize();
76   glutMainLoop();
77   return 0;
78 }
79
```

```
 1 /*
 2 ** double.c - ダブルバッファでアニメーションを行なう
 3 */
 4
 5 #include <GL/glut.h>
 6
 7 void
 8 initialize(void)
 9 {
10   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
11 }
12
13 void
14 display(void)
15 {
16   static float offsetX = -1.0;
17
18   offsetX += 1.0;
19   if( offsetX>=100.0 ) offsetX -= 100.0;
20
21   glClear( GL_COLOR_BUFFER_BIT );
22
23   glColor3f( 1.0f, 1.0f, 0.0f );
24
25   glRectf( -80.0f+offsetX, -80.0f, -20.0f+offsetX, -20.0f );
26
27   /* バックバッファとフロントバッファを入れ換える */
28   glutSwapBuffers();
29
30   /* glFlush()は、glutSwapBuffers()内で行なわれるため不要 */
31   /*
32     glFlush();
33   */
34 }
35
36 void
37 keyboard( unsigned char c, int x, int y )
38 {
39   if( c==27 ) exit( 0 );
40 }
41
42 void
43 reshape( int w, int h )
```

```
44 {
45   glViewport( 0, 0, w, h );
46
47   glMatrixMode( GL_PROJECTION );
48   glLoadIdentity();
49   gluOrtho2D( -100.0, 100.0, -100.0, 100.0 );
50   glMatrixMode( GL_MODELVIEW );
51 }
52
53 void
54 animate(void)
55 {
56   glutPostRedisplay();
57 }
58
59 int
60 main( int argc, char *argv[] )
61 {
62   glutInit( &argc, argv );
63
64   /* ダブルバッファの確保 */
65   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE );
66
67   glutInitWindowPosition( 100, 100 );
68   glutInitWindowSize( 500, 400 );
69   glutCreateWindow( argv[0] );
70
71   glutDisplayFunc( display );
72   glutKeyboardFunc( keyboard );
73   glutReshapeFunc( reshape );
74
75   glutIdleFunc( animate );
76
77   initialize();
78   glutMainLoop();
79   return 0;
80 }
81
```

<演習>

1. forYou ディレクトリの reshape プログラムを実行して下さい。

```
% cd forYou
% make reshape
% reshape
```

   さらにウィンドウ形状を変化させ、絵が Window 形状の変化に追従してマッピングされていることを確認して下さい。
2. reshape.c を作業ディレクトリにコピーしてから、reshape 関数を書き換えて、Window サイズを変えても図形の縦横比が保存されるようにして下さい。縦が長い場合と横が長い場合に分けて、glViewport 関数でビューポートの形状を設定します。
   (この例題は時間に余裕のある方が取り組んで下さい)

```
% cd ~
% cp forYou/reshape.c bobby/reshape.c
% cd bobby
% vi reshape.c
...（適当にプログラムの変更を行なう）
% make reshape
% reshape
```

   さらにウィンドウ形状を変化させ、絵が崩れないことを確認して下さい。
3. forYou ディレクトリの single と double プログラムを実行して下さい。

```
% cd forYou
% make single double
% single
% double
```

4. double.c を作業ディレクトリの animation.c にコピーして、色の異なる長方形をいくつか追加して下さい。

```
% cd ~
% cp forYou/double.c bobby/animation.c
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

サブルーチンのまとめ

GLUT 関数

```
void glutInit(int *argcp, char **argv);
void glutInitWindowPosition(int x0, int y0);
void glutInitWindowSize(int width, int height);
void glutInitDisplayMode(GLenum mode);
int  glutCreateWindow(char *title);
void glutMainLoop(void);
void glutSwapBuffers(void);
void glutIdleFunc(void (*func)());
void glutReshapeFunc(void (*func)(int width, int height));
```

初期設定など

```
void glClearColor(GLclampf red, GLclampf green, GLclampf blue,
                  GLclampf alpha);
void glClear(GLbitfield mask);

void gluOrtho2D(GLdouble left, GLdouble right,
                GLdouble bottom, GLdouble top);
void glColor3f(GLfloat red, GLfloat green, GLfloat blue);
void glClear( void );
void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);
```

描画ルーチン

```
void glRectf(GLfloat x0, GLfloat y0, GLfloat x1, GLfloat y1);
```

第2章　描画

本章の目的

- OpenGL の各種図形プリミティブの種類を知る
- 頂点ルーチンの使い方を学び、2 次元物体、3 次元物体の描画方法を理解する
- 独立ポリゴンとメッシュの違いを理解する
- 各種図形プリミティブ毎の描画速度の違いを把握する

2.1　点、線、ポリゴン・プリミティブの描画

- 点、線、ポリゴン（多角形）の描画は全く同じ関数を用いる
- 3 種類の関数を組み合わせ次のシーケンスを使用

```
glBegin(プリミティブ名);
glVertex*(...);  /* 頂点座標の指定 */
...
glEnd();
```

- void glBegin(GLenum mode);
  glBegin に続いて頂点ルーチンでプリミティブを構成する頂点座標を指定していく
- glBegin の引数 mode には次のものを指定することができる
  (各 mode の使い方は後述する)
  - 点
    GL_POINTS
  - 線
    GL_LINES
    GL_LINE_STRIP
    GL_LINE_LOOP
  - ポリゴン
    GL_POLYGON
    GL_TRIANGLES
    GL_QUADS
    GL_TRIANGLE_STRIP
    GL_TRIANGLE_FAN
    GL_QUAD_STRIP
- void glEnd(void);
  頂点リストの指定を終え描画シーケンスを終了する

頂点座標の指定 (glVertex*関数)

- 頂点の指定
- 精度、次元などで 24 種類の頂点ルーチンがある[1]
- void glVertex{234}{sifd}[v](TYPE coords);
  {} は括弧の中のもの 1 つを選ぶ、[] はあってもなくてもよい

  2 : x 座標と y 座標を指定（z 座標は 0）
  3 : x 座標、y 座標、z 座標を指定
  4 : 同次座標形で x 座標、y 座標、z 座標、w 座標を指定

  s : short の s、2 byte 整数
  i : integer の i、4 byte 整数
  f : float の f、4 byte 浮動小数点
  d : double の d、8 byte 浮動小数点

  v : 頂点座標値を格納した配列の先頭アドレスを引数で渡す
  v なし : 頂点座標値を引数で列挙して渡す

- 引数に座標値を列挙して指定する場合

```
glVertex2f(1.0f, 2.0f);
glVertex3d(-1.0, -3.0, 1.0);
```

- 座標値を格納した配列の先頭アドレスで指定する場合

```
int array3[3] = {5, 8, 7};
float array4[4] = {5.0f, 8.0f, 7.0f, 2.0f};
...
glVertex3iv(array3);
glVertex4fv(array4);
```

- OpenGL の関数名、定数名、タイプ名に関する規約は付録 D にまとめた

[1] glVertex*は 1 つの呼び出しで 1 つの頂点を指定する
ただし複数頂点を 1 回の glVertex*関数呼び出しで行う方法も OpenGL 1.1 からサポートされるようになった

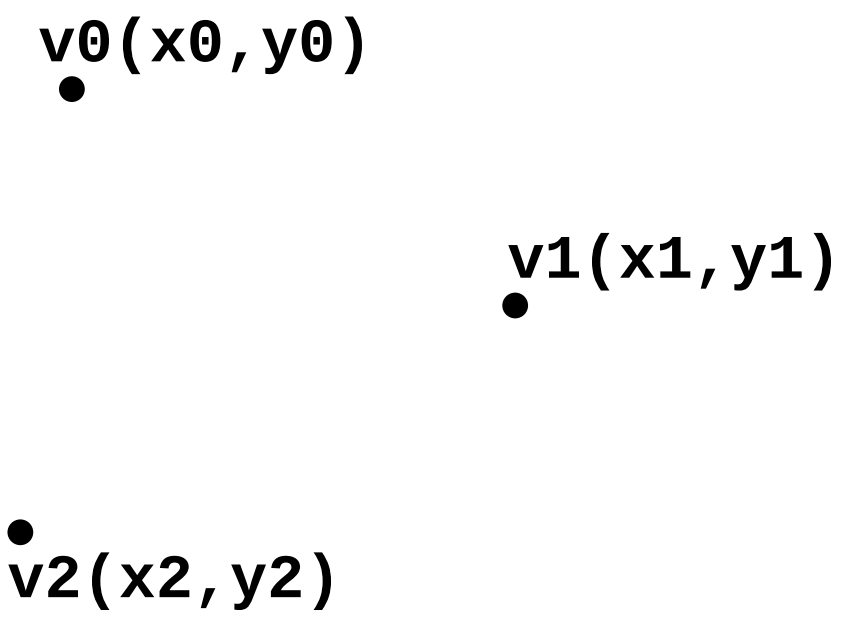
頂点座標の指定 (glVertex*関数) – 続き –

- 1 つの描画シーケンス (glBegin—glEnd に囲まれた間) の中に異なる次元、異なる型、異なる引数タイプの頂点関数を混在できる

```
int array3[3] = {5, 8, 7};
float array4[4] = {5.0f, 8.0f, 7.0f, 2.0f};

glBegin(GL_LINE_STRIP);
glVertex2f(1.0f, 2.0f);
glVertex3d(-1.0, -3.0, 1.0);
glVertex3iv(array3);
glVertex4fv(array4);
glEnd();
```

- 2 次元で指定した頂点は z 座標を 0 とした 3 次元座標に等しい
  glVertex2f(x, y); と glVertex3f(x, y, 0.0f); は同じ座標を指定している
- 4 次元の同次座標系で指定した座標は、は 4 次の要素 w で x、y、z を割ると 3 次元の座標値に変換できる
  glVertex4f(x, y, z, w); と glVertex3f(x/w, y/w, z/w); は同じ座標値を指定することになる

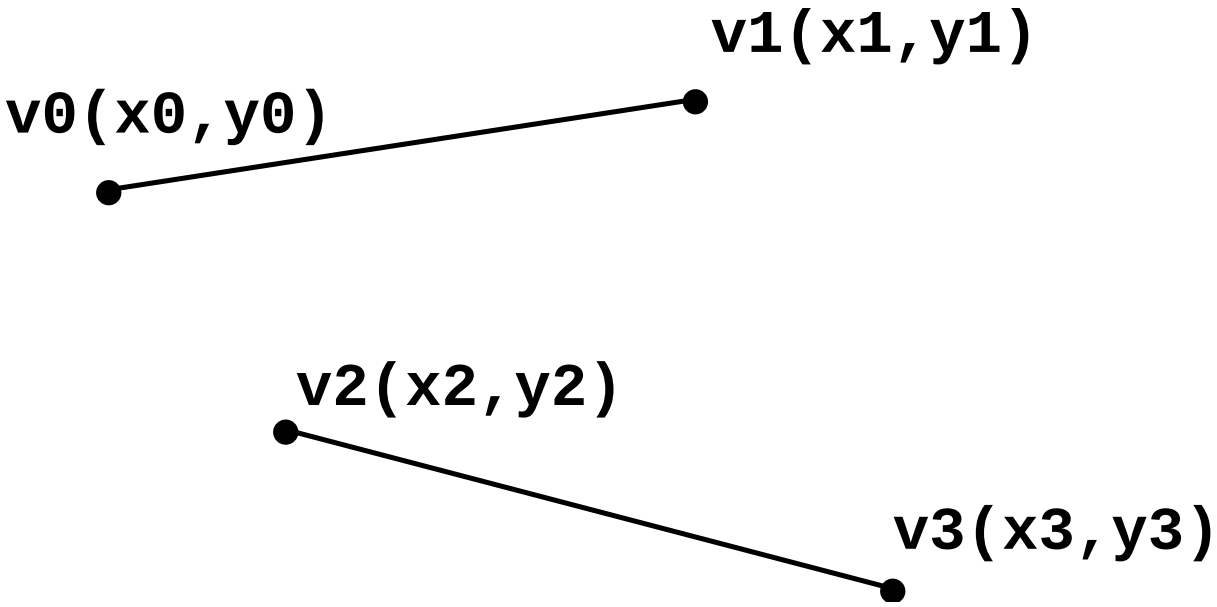
点の描画



- 点の描画

```
float v0[2] = {x0,y0};
float v1[2] = {x1,y1};
float v2[2] = {x2,y2};
...
glBegin(GL_POINTS);
glVertex2fv(v0);
glVertex2fv(v1);
glVertex2fv(v2);
glEnd();
...
```

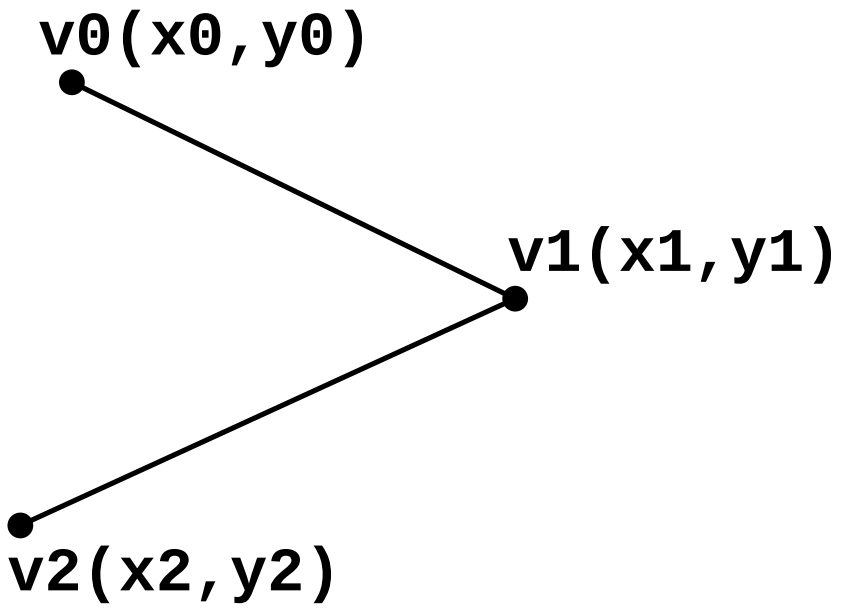
線分の描画



- 線分の描画

```
float v0[2] = {x0,y0};
float v1[2] = {x1,y1};
float v2[2] = {x2,y2};
float v3[2] = {x3,y3};
...
glBegin(GL_LINES);
glVertex2fv(v0);
glVertex2fv(v1);
glVertex2fv(v2);
glVertex2fv(v3);
glEnd();
...
```

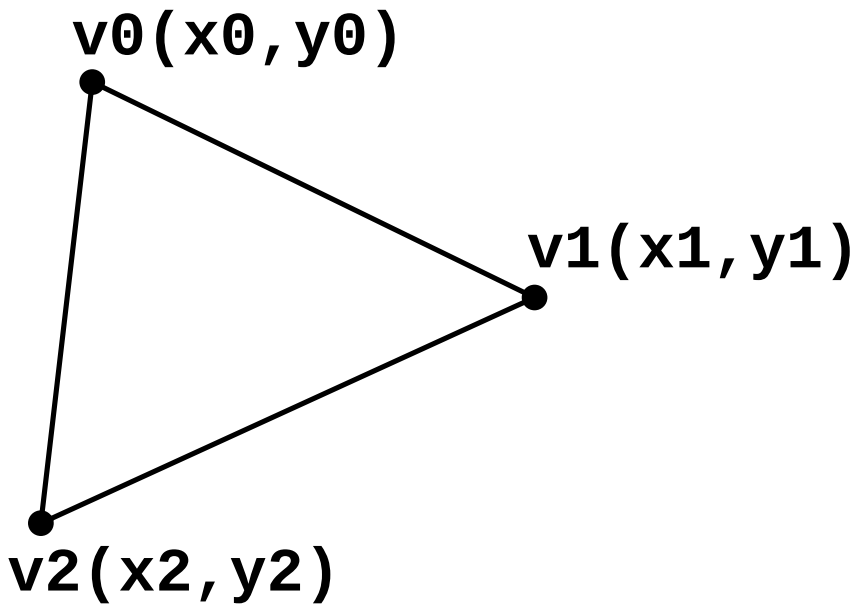
連続する線の描画



- 連続する線の描画

```
float v0[2] = {x0,y0};
float v1[2] = {x1,y1};
float v2[2] = {x2,y2};
...
glBegin(GL_LINE_STRIP);
glVertex2fv(v0);
glVertex2fv(v1);
glVertex2fv(v2);
glEnd();
...
```

線の描画（ワイヤーフレームの多角形）



- 使い方は GL_LINE_STRIP と同じ

```
float v0[2] = {x0,y0};
float v1[2] = {x1,y1};
float v2[2] = {x2,y2};
...
glBegin(GL_LINE_LOOP);
glVertex2fv(v0);
glVertex2fv(v1);
glVertex2fv(v2);
glEnd();
...
```

- ワイヤーフレームの多角形は glPolygonMode 関数とポリゴン描画関数を組み合わせて使う方法もある

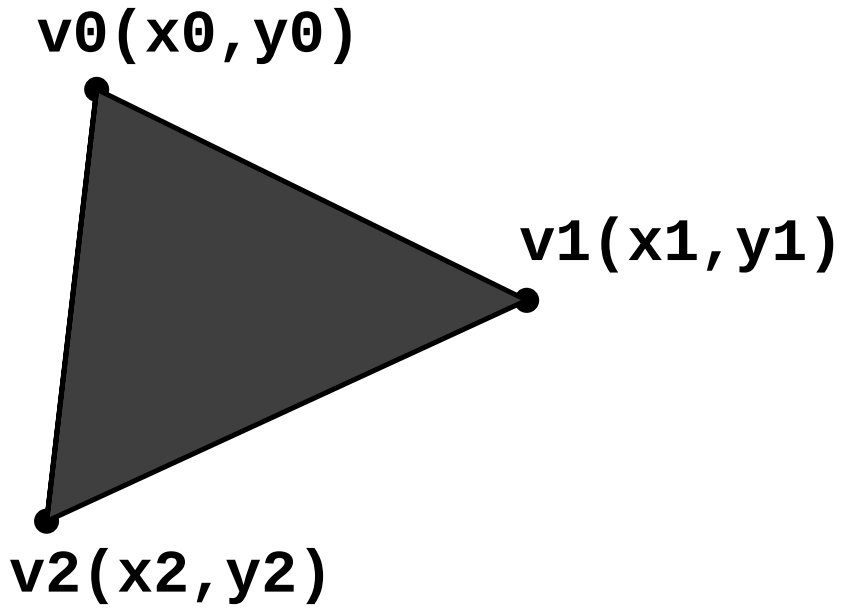
```
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
glBegin(GL_POLYGON);
glVertex2fv(v0);
glVertex2fv(v1);
glVertex2fv(v2);
glEnd();
```

塗りつぶし多角形の描画

- OpenGL では凸状のねじれのないポリゴンのみ正確な塗りつぶしができる
- 凹状のポリゴン、穴の空いたポリゴンは GLU ライブラリで三角形分解してから塗りつぶす
- 塗りつぶし多角形を描く方法は次の 2 つ
  - 独立ポリゴンの描画モードを使う
    GL_POLYGON, GL_QUADS, GL_TRIANGLES
  - メッシュ領域の描画モードを使う
    GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUAD_STRIP
- 処理速度はほぼ次のようになる
  GL_TRIANGLE_STRIP > GL_QUAD_STRIP
  > GL_QUADS, GL_TRIANGLES > GL_POLYGON

独立ポリゴンの描画 – GL_POLYGON –

- ポリゴンの塗りつぶし方法としては効率は悪い
- 面ごとに各頂点の法線を設定できるので、階段などの表現ができる

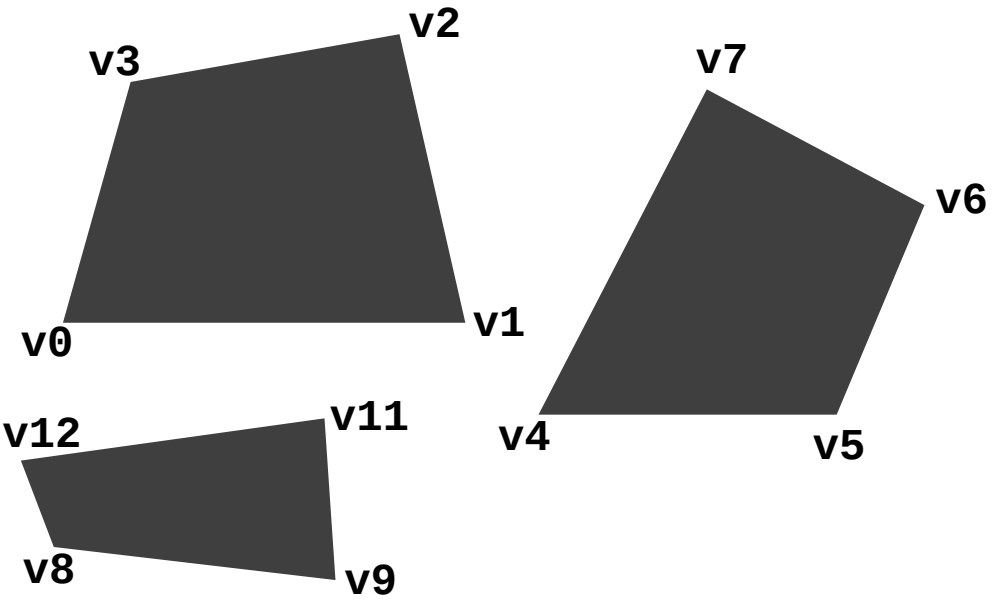


- コーディング例

```
...
float v0[2] = {x0,y0};
float v1[2] = {x1,y1};
float v2[2] = {x2,y2};
...
glBegin(GL_POLYGON);
glVertex2fv(v0);
glVertex2fv(v1);
glVertex2fv(v2);
glEnd();
...
```

複数独立ポリゴンの描画
– GL_QUADS, GL_TRIANGLES –

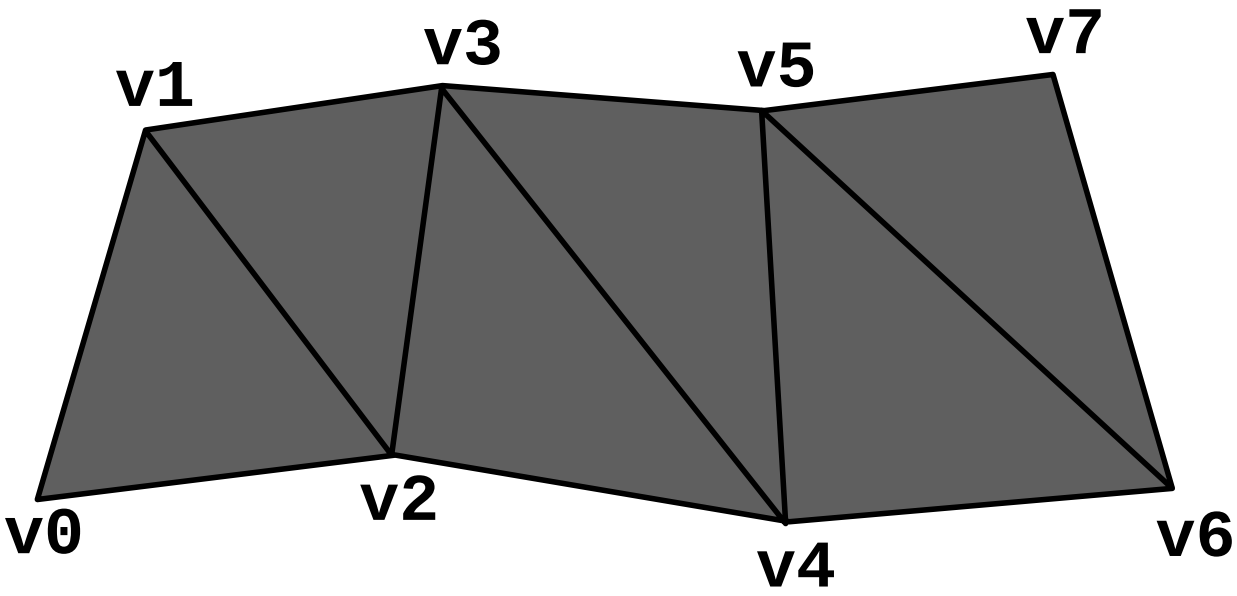
- 複数の三角形、あるいは複数の四角形をまとめて描画することができる
- 三角形と四角形を混在することはできない
- 多数の独立ポリゴンを描画するときはできるだけ GL_QUADS または GL_TRIANGLES を使う
- GL_POLYGON を何回も繰り返すよりも高速 (OpenGL チューニングのポイント)



- コーディング例

```
...
glBegin(GL_QUADS);
glVertex2fv(v0); glVertex2fv(v1);
glVertex2fv(v2); glVertex2fv(v3);
glVertex2fv(v4); glVertex2fv(v5);
glVertex2fv(v6); glVertex2fv(v7);
glVertex2fv(v8); glVertex2fv(v9);
glVertex2fv(v10); glVertex2fv(v11);
glEnd();
...
```

メッシュによるポリゴン塗りつぶし 1
– GL_TRIANGLE_STRIP –



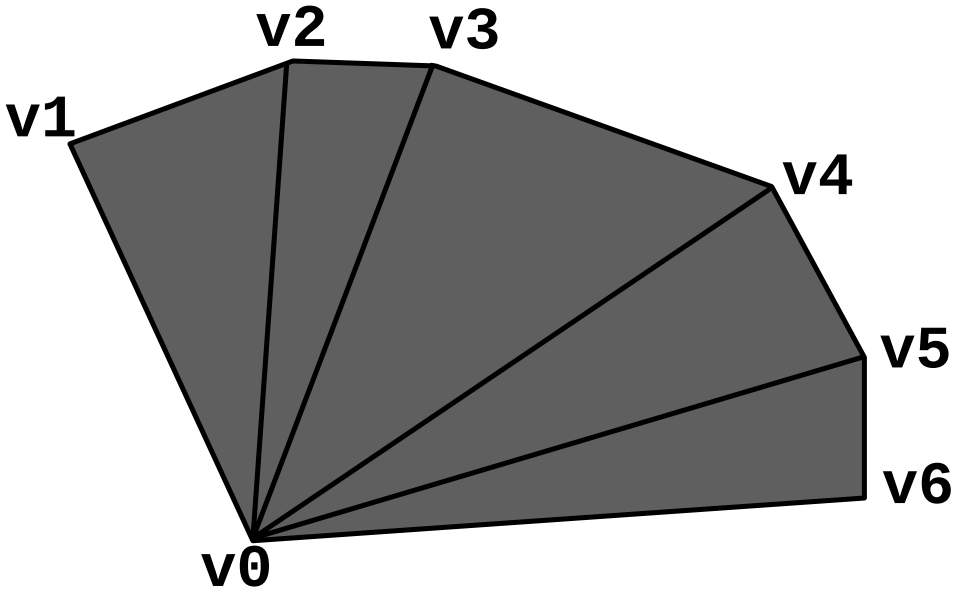
- コーディング例

```
...
glBegin(GL_TRIANGLE_STRIP);
glVertex2fv(v0); glVertex2fv(v1);
glVertex2fv(v2); glVertex2fv(v3);
glVertex2fv(v4); glVertex2fv(v5);
glVertex2fv(v6); glVertex2fv(v7);
glEnd();
...
```

注：上図の各ポリゴンの輪郭線は見易さのためで、実際の出力にはない

メッシュによるポリゴン塗りつぶし 2
– GL_TRIANGLE_FAN –

- 1 点を中心とした扇状の領域を塗りつぶす
- 中心となる点の法線が 1 つの方向に限定されるため、円錐の描画には向かない
  中心点で微分可能な球の底のような部分の描画に向く



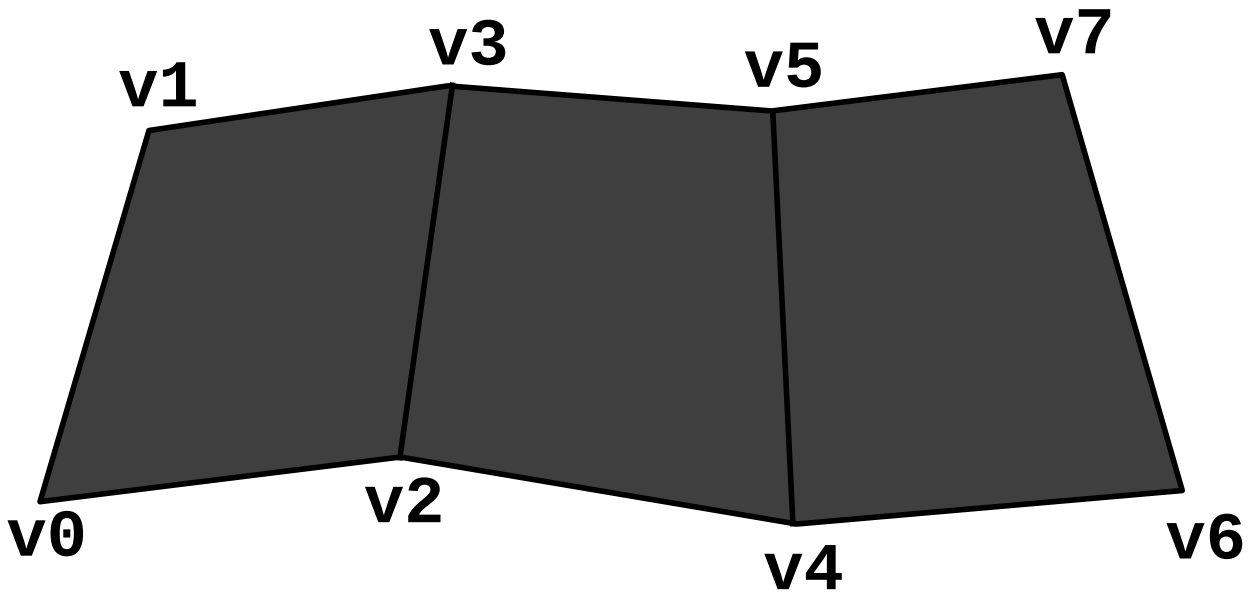
- コーディング例

```
...
glBegin(GL_TRIANGLE_FAN);
glVertex2fv(v0); /* 中心の点 */
glVertex2fv(v1); glVertex2fv(v2);
glVertex2fv(v3); glVertex2fv(v4);
glVertex2fv(v5); glVertex2fv(v6);
glEnd();
...
```

注：上図の各ポリゴンの輪郭線は見易さのためで、実際の出力にはない

メッシュによるポリゴン塗りつぶし 3
– GL_QUAD_STRIP –

- 三角形メッシュ(GL_TRIANGLE_*) に次いで高速
- 法線ベクトルの処理の関係で円錐、円筒などの描画が三角形メッシュより美しい



- コーディング例

```
...
glBegin(GL_QUAD_STRIP);
glVertex2fv(v0); glVertex2fv(v1);
glVertex2fv(v2); glVertex2fv(v3);
glVertex2fv(v4); glVertex2fv(v5);
glVertex2fv(v6); glVertex2fv(v7);
glEnd();
...
```

注：上図の各ポリゴンの輪郭線は見易さのためで、実際の出力にはない

glBegin()、glEnd() の使い方

- glBegin() と glEnd() で挟まれた間で使うことのできる OpenGL 関数は次の通り
  - glVertex*() ：頂点座標の指定
  - glColor*() ：RGB カラーの設定
  - glIndex*() ：カラーインデックスの設定
  - glNormal*() ：法線ベクターの設定
  - glEvalCoord*() : 座標の生成
  - glCallList*() ：ディスプレイリストの呼び出し
  - glTexCoord*() : テクスチャ座標の指定
  - glEdgeFlag*() : 線を書くか否かの制御
  - glMaterial*() ：材料特性の設定

- glColor*()、glNormal*()、glTexCoord*() は glVertex*() と組み合わせ、各頂点毎の色、法線ベクター、テクスチャ座標を指定するのに用いる

- コーディング例
  赤、黄色、緑を各頂点に割り振りグーロー・シェイディングで三角形を塗りつぶす

```
glBegin(GL_POLYGON);
glColor3f(1.0, 0.0, 0.0);
glVertex2f(-0.7, -0.5);

glColor3f(1.0, 1.0, 0.0);
glVertex2f(0.7, -0.3);

glColor3f(0.0, 1.0, 0.0);
glVertex2f(-0.1, 0.7);
glEnd();
```

```
 1 /*
 2 ** primitive.c - glBegin()、glEnd() を使った描画
 3 */
 4
 5 #include <GL/glut.h>
 6
 7 void
 8 initialize(void)
 9 {
10   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
11 }
12
13 /* 三角形の描画 */
14 void
15 drawTri( float offset )
16 {
17   /* プリミティブ描画関数による三角形の描画 */
18   glBegin( GL_TRIANGLES );
19   /* 赤い頂点 */
20   glColor3f( 1.0f, 0.0f, 0.0f );
21   glVertex2f( -80.0f+offset, -80.0f);
22
23   /* 緑の頂点 */
24   glColor3f( 0.0f, 1.0f, 0.0f );
25   glVertex2f( -20.0+offset, -70.0);
26
27   /* 青い頂点 */
28   glColor3f( 0.0f, 0.0f, 1.0f );
29   glVertex2f( -40.0f+offset, -20.0f);
30   glEnd();
31 }
32
33 void
34 display(void)
35 {
36   static float offsetX = -1.0;
37
38   offsetX += 1.0;
39   if( offsetX>=100.0 ) offsetX -= 100.0;
40
41   glClear( GL_COLOR_BUFFER_BIT );
42
43   glColor3f( 1.0f, 1.0f, 0.0f );
```

```
44
45   /* offsetX分ずれた三角形を描画 */
46   drawTri( offsetX );
47
48   glutSwapBuffers();
49 }
50
51 void
52 keyboard( unsigned char c, int x, int y )
53 {
54   if( c==27 ) exit( 0 );
55 }
56
57 void
58 reshape( int w, int h )
59 {
60   glViewport( 0, 0, w, h );
61
62   glMatrixMode( GL_PROJECTION );
63   glLoadIdentity();
64   gluOrtho2D( -100.0, 100.0, -100.0, 100.0 );
65   glMatrixMode( GL_MODELVIEW );
66 }
67
68 void
69 animate(void)
70 {
71   glutPostRedisplay();
72 }
73
74 int
75 main( int argc, char *argv[] )
76 {
77   glutInit( &argc, argv );
78
79   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE );
80   glutInitWindowPosition( 100, 100 );
81   glutInitWindowSize( 500, 400 );
82   glutCreateWindow( argv[0] );
83
84   glutDisplayFunc( display );
85   glutKeyboardFunc( keyboard );
86   glutReshapeFunc( reshape );
87   glutIdleFunc( animate );
```

```
 88
 89   initialize();
 90   glutMainLoop();
 91   return 0;
 92 }
 93
```

プリミティブ属性の変更

- 点、線、ポリゴンの各プリミティブは描画属性を変更することができる
- 点の大きさを指定する
  - void glPointSize(GLfloat size);
  - ピクセル単位で size x size の四角形になる
    デフォルトは 1.0f
- 線の太さを指定する
  - void glLineWidth(GLfloat width);
  - ピクセル単位で線の太さを指定する
    デフォルトは 1.0f
- 線のパターンを指定する
  - void glLineStipple(GLint factor, GLushort pattern);
  - factor はパターンの倍率
  - pattern がビットのオン/オフで示した線の描画パターン
  - パターンによる描画は glEnable 関数を GL_LINE_STIPPLE 定数を引数にして呼び出すと有効になる
  - パターンによる描画をオフにするには GL_LINE_STIPPLE 定数を引数にして glDisable を呼び出す
- コーディング例

```
glEnable(GL_LINE_STIPPLE);
glLineStipple(3, 0xf0f0); /* dashed line */
glBegin(GL_LINE_STRIP);
glVertex2f(-0.7, -0.5);
glVertex2f(0.7, -0.3);
glVertex2f(-0.1, 0.7);
glEnd();
```

ポリゴン描画属性の変更

- ポリゴン描画は詳細な設定ができる
- ポリゴンの表 (front face) と裏 (back face)
  次のコーリング・シーケンスでポリゴンを描画した場合頂点の描画順がウィンドウに対して時計回りのものをバックフェース・ポリゴン、半時計回りのものをフロントフェース・ポリゴンと呼ぶ

```
glBegin(GL_POLYGON);
glVertex2fv(v0);
glVertex2fv(v1);
glVertex2fv(v2);
glVertex2fv(v3);
glEnd();
```

- より正確にはポリゴンのウィンドウ座標系に対し次の計算を実行して決定される

$$a = \frac{1}{2} + \sum_{i=0}^{n-1} (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i)$$

- 描画モードの設定
  void glPolygonMode(GLenum face, GLenum mode);
  – ポリゴン描画について、塗りつぶし、ワイヤーフレーム、頂点のみ、をフロントフェース、バックフェース、あるいは両方について指定することができる
  – face は GL_FRONT、GL_BACK、GL_FRONT_AND_BACK の中から選ぶ
  – mode は GL_FILL、GL_LINE、GL_POINT から選ぶ
  – デフォルトは GL_FRONT_AND_BACK と GL_FILL の組み合わせ
- パターンを設定しての塗りつぶし
  void glPolygonStipple(const GLubyte *mask);
  – 32x32 の大きさのビット・パターンを使ってポリゴンを塗りつぶすことができる
  – パターンによる塗りつぶしは GL_POLYGON_STIPPLE 定数を引数に glEnable を呼び出して有効にする。無効にするには同様に glDisable を呼び出す

```
 1 /*
 2 ** stipple.c - glLineStipple() を使った描画
 3 */
 4
 5 #include <GL/glut.h>
 6
 7 void
 8 initialize(void)
 9 {
10   /* ラインパターンの初期化 */
11   glLineWidth( 5.0 );
12   glLineStipple( 5, 0xf0f0 );
13   glPolygonMode( GL_FRONT_AND_BACK, GL_LINE );
14   glEnable( GL_LINE_STIPPLE );
15
16   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
17 }
18
19 void
20 drawTri( float offset )
21 {
22   glBegin( GL_TRIANGLES );
23   glColor3f( 1.0f, 0.0f, 0.0f );
24   glVertex2f( -80.0f+offset, -80.0f);
25   glColor3f( 0.0f, 1.0f, 0.0f );
26   glVertex2f( -20.0+offset, -70.0);
27   glColor3f( 0.0f, 0.0f, 1.0f );
28   glVertex2f( -40.0f+offset, -20.0f);
29   glEnd();
30 }
31
32 void
33 display(void)
34 {
35   static float offsetX = -1.0f;
36
37   offsetX += 1.0f;
38   if( offsetX>=100.0f ) offsetX -= 100.0f;
39
40   glClear( GL_COLOR_BUFFER_BIT );
41
42   glColor3f( 1.0f, 1.0f, 0.0f );
43
```

```
44   drawTri( offsetX );
45
46   glutSwapBuffers();
47 }
48
49 void
50 keyboard( unsigned char c, int x, int y )
51 {
52   if( c==27 ) exit( 0 );
53 }
54
55 void
56 reshape( int w, int h )
57 {
58   glViewport( 0, 0, w, h );
59
60   glMatrixMode( GL_PROJECTION );
61   glLoadIdentity();
62   gluOrtho2D( -100.0, 100.0, -100.0, 100.0 );
63   glMatrixMode( GL_MODELVIEW );
64 }
65
66 void
67 animate(void)
68 {
69   glutPostRedisplay();
70 }
71
72 int
73 main( int argc, char *argv[] )
74 {
75   glutInit( &argc, argv );
76
77   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE );
78   glutInitWindowPosition( 100, 100 );
79   glutInitWindowSize( 500, 400 );
80   glutCreateWindow( argv[0] );
81
82   glutDisplayFunc( display );
83   glutKeyboardFunc( keyboard );
84   glutReshapeFunc( reshape );
85   glutIdleFunc( animate );
86
```

```
 87   initialize();
 88   glutMainLoop();
 89   return 0;
 90 }
 91
```

<演習>

1. forYou ディレクトリの primitive、stipple プログラムを実行して下さい。

```
% cd forYou
% make primitive
% primitive
% make stipple
% stipple
```

2. 作業ディレクトリの animation プログラムに変更を加え、glBegin() などを用いた自由な描画を行なって下さい。雪だるま、海辺の風景など、何かテーマを決めるとやりやすいでしょう。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

サブルーチンのまとめ

描画ルーチン

```
void glBegin(GLenum mode);
void glEnd();
```

頂点ルーチン

```
void glVertex2d (GLdouble x, GLdouble y);
void glVertex2dv (const GLdouble *v);
void glVertex2f (GLfloat x, GLfloat y);
void glVertex2fv (const GLfloat *v);
void glVertex2i (GLint x, GLint y);
void glVertex2iv (const GLint *v);
void glVertex2s (GLshort x, GLshort y);
void glVertex2sv (const GLshort *v);
void glVertex3d (GLdouble x, GLdouble y, GLdouble z);
void glVertex3dv (const GLdouble *v);
void glVertex3f (GLfloat x, GLfloat y, GLfloat z);
void glVertex3fv (const GLfloat *v);
void glVertex3i (GLint x, GLint y, GLint z);
void glVertex3iv (const GLint *v);
void glVertex3s (GLshort x, GLshort y, GLshort z);
void glVertex3sv (const GLshort *v);
```

プリミティブ属性変更関数

```
void glLineStipple(GLint factor, GLushort pattern);
void glLineWidth(GLfloat width);
void glPointSize(GLfloat size);
void glPolygonMode(GLenum face, GLenum mode);
void glPolygonStipple(const GLubyte *mask);
```

法線ベクターの指定

```
void glNormal*(...);
```

テクスチャ座標の指定

```
void glTexCoord*(...);
```

第3章　3次元グラフィックスの基礎

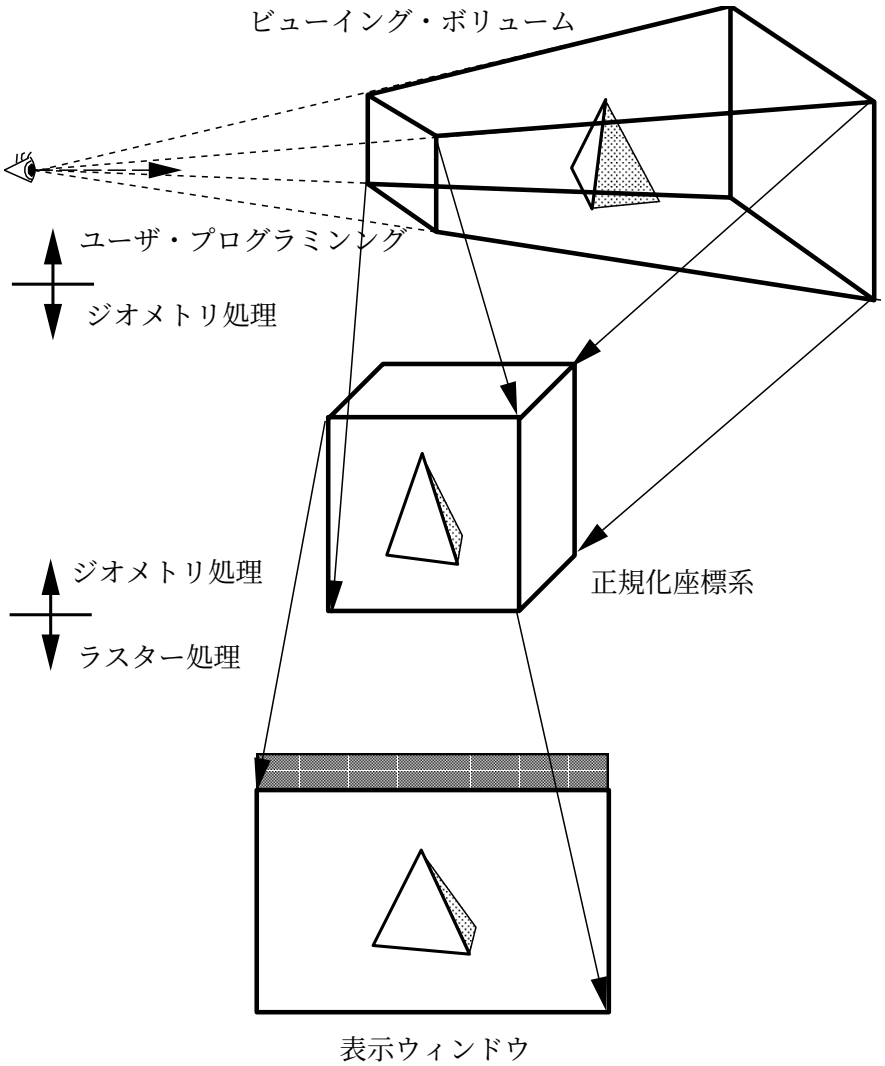
本章の目的

- 3 次元グラフィックス全体の処理の流れを理解する
- 透視投影変換、平行投影変換の効果とその使用法
- ビューイング変換の使用法
- 3D モデルを使ったプログラムの作成

3.1　3次元グラフィックス概要

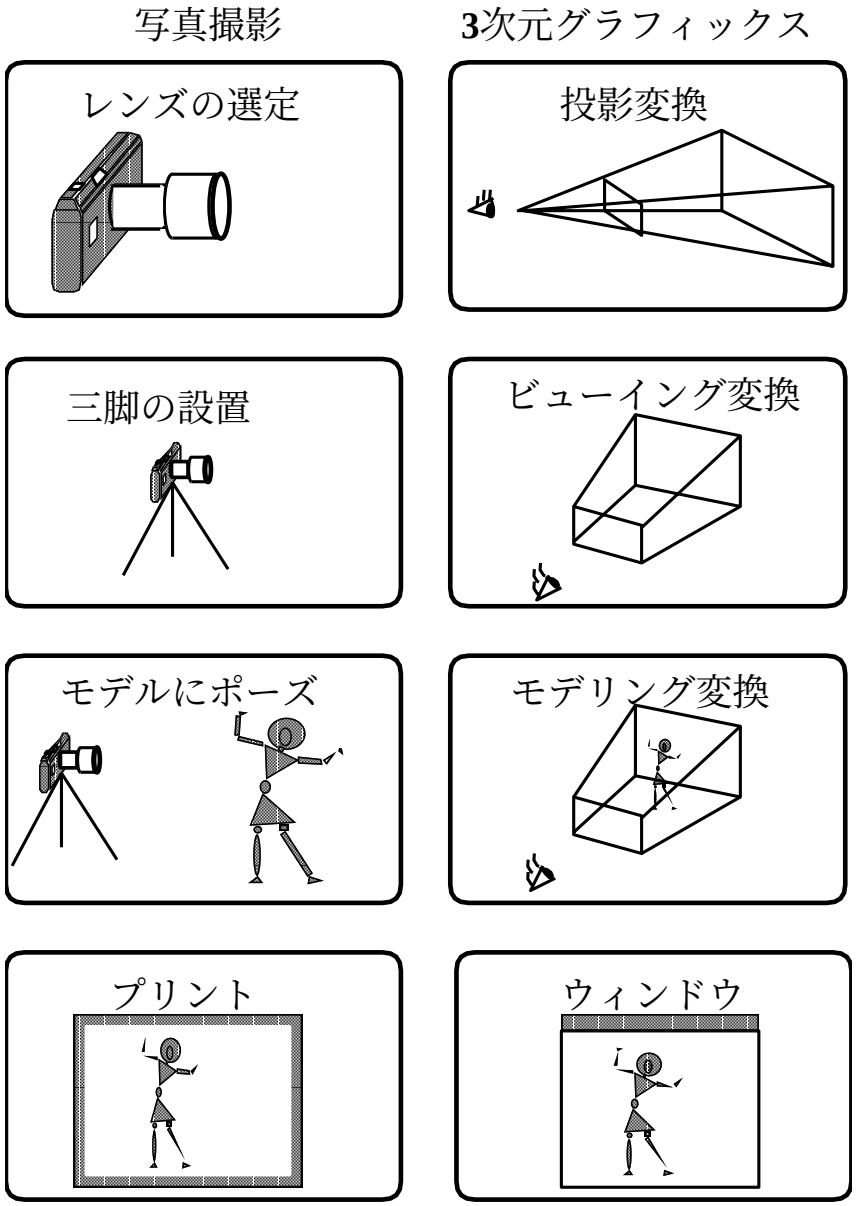
3次元グラフィックスの仕組み

- 3次元シーンの描画の仕組みは次のようになっている
  1. ユーザはワールド空間中にビューイング・ボリューム (可視領域) を設定しそこに3次元シーンを描画する
  2. OpenGL はビューイング・ボリュームの外にはみ出した3次元シーンをクリッピングし、ビューイング・ボリューム内部のシーンを正規化座標系にマッピングする
     この段階で各頂点の色を決定する
  3. OpenGL は続いて、正規化座標系からウィンドウのビューポートにマッピングが行ない、各頂点の色から各面を塗りつぶす



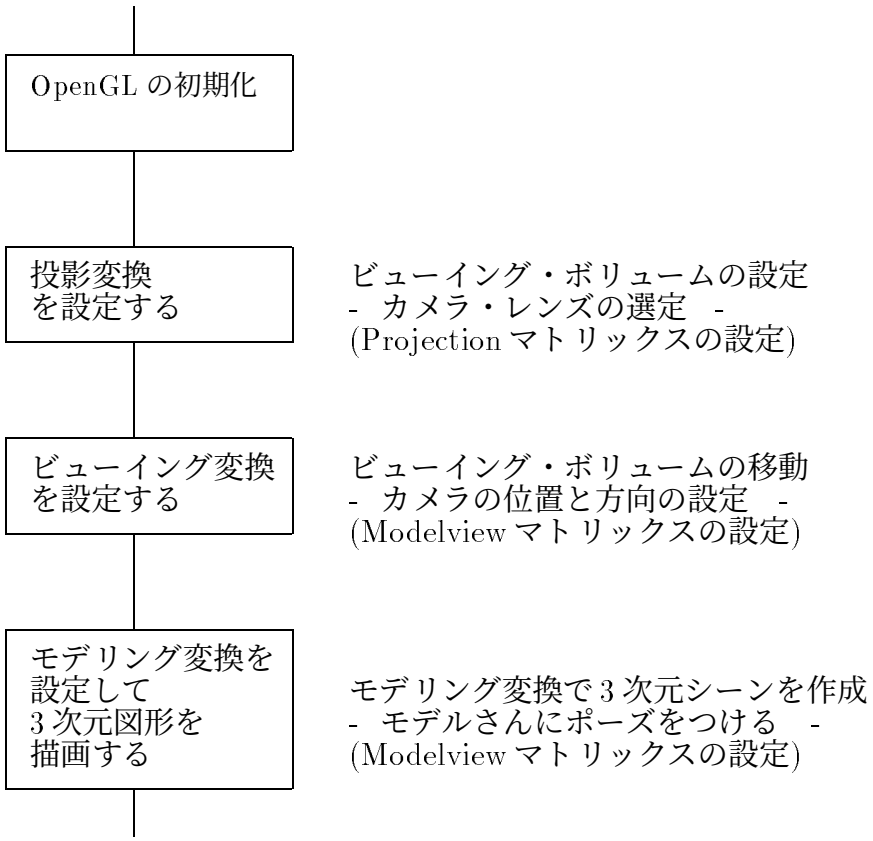
3次元グラフィックスと写真撮影の対応

- ビューイング・ボリュームの設定から 3 次元シーンの描画までの部分を写真撮影になぞらえると次のようになる



3次元グラフィックス・プログラミングの流れ

- 3次元グラフィックスのプログラミングの流れを示すと次のようになる

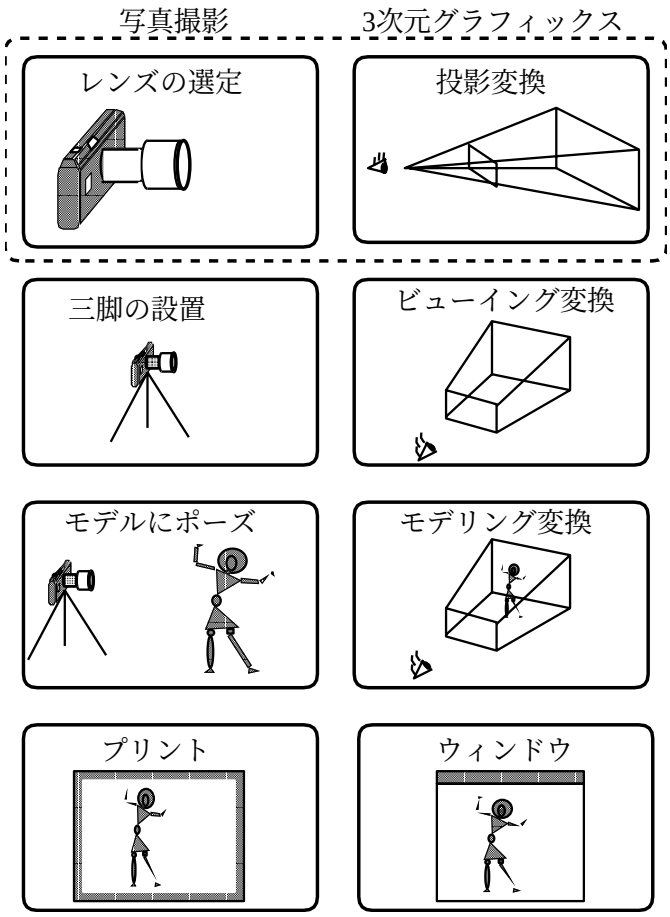


- ビューイング・ボリューム（描画領域）の形状と大きさを設定して投影変換の定義を行う
- ビューイング変換によりビューイング・ボリュームをワールド空間中の適当な位置に配置する
- モデリング変換で物体を適当な形状、大きさ、位置になるように設定して描画する
- 投影変換、ビューイング変換を本章で、モデリング変換を次章で説明する

3.2 投影変換の設定

投影変換と写真撮影の対応

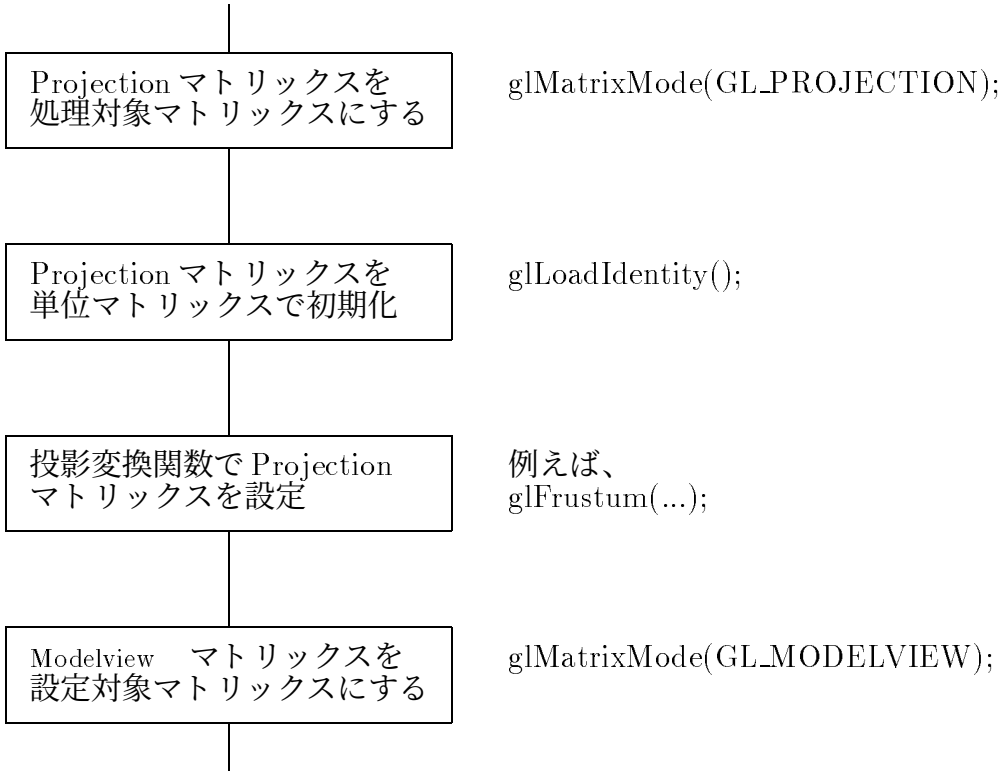
- 投影変換の役割を前出の写真撮影の図で再確認する



- ビューイング・ボリュームの形状と大きさを定める投影変換は写真撮影におけるレンズの選択に相当する
- 例えば、広角レンズや望遠レンズを選んだり、倍率を決めたりするのに等しい

投影変換の設定手順

- 投影変換の設定は3つのステップに分かれる
  1. 処理対象マトリックスを投影変換を担当するProjectionマトリックスにする
  2. Projectionマトリックスを単位行列で初期化する
  3. glFrustumやglOrthoなどの投影変換設定関数でビューイング・ボリュームの形状と大きさを指定する



- 以上によりビューイング・ボリューム内の座標を正規化座標系に投影するProjectionマトリックスが設定される

投影変換設定の前準備

投影変換を設定する前に、処理対象マトリックスを Projection マトリックスに切替えて単位行列で初期化する

- カレントの処理対象マトリックスを切替える
  void glMatrixMode (GLenum mode);
  - OpenGL では Projection、Modelvew、Texture の 3 つの変換マトリックスがある
  - マトリックス処理関数は 3 つの変換マトリックスの中からカレントの処理対象マトリックスとして選択されているものに作用する
    単位行列初期化関数、投影変換設定関数、モデリング変換設定関数などを実行する前には適宜、処理対象マトリックスの選択を行う
  - それぞれ GL_PROJECTION、GL_MODELVIEW、GL_TEXTURE の各定数を引数として glMatrixMode を呼ぶことによりカレントの処理対象マトリックスとすることができる
  - Projection マトリックスを操作対象マトリックスに設定する場合は次の通り
    ```
    glMatrixMode(GL_PROJECTION);
    ```
- カレントの処理対象マトリックスを単位行列で初期化する
  void glLoadIdentity (void);
  - glLoadIdentity は glMatrixMode で指定されたカレントの処理対象マトリックスを単位行列で置き換える
  - カレントの変換マトリックスに対して置換えを行うマトリックス操作関数は glLoadIdentity と glLoadMatrix の 2 つの関数だけである
  - その他の 3 次元の各種変換を設定する関数は既存のマトリックスに対し掛け合わせを行う
    投影変換やモデリング変換の設定関数も掛け合わせを行う
  - プログラムの始めに Modelview、Projection の各マトリックスを glLoadIdentity 関数で初期化しておく

投影変換の設定

投影変換は大きく分けると2種類のものがあり、各々ビューイング・ボリュームの形状が異なる

透視投影変換

ビューイング・ボリュームの形状は踏台の形、あるいはそれをななめに押し潰した（剪断変形させた）形になる
透視投影の特徴は次の通り

- 目に近い物体が大きく、遠い物体が小さく変換されるため、遠近感がはっきりした自然な3次元画像が得られる
- 隠面処理を行うときには、ビューイング・ボリュームのnear面とfar面をできるだけコンパクトに設定する[1]
- 透視投影変換をマトリックスとベクターのかけ算で表現するために3次元グラフィックスでは同次座標系で4次元のマトリックス演算を用いる
- 透視投影の設定関数は次の2種類がある
  - glFrustum
  - gluPerspective

平行投影変換

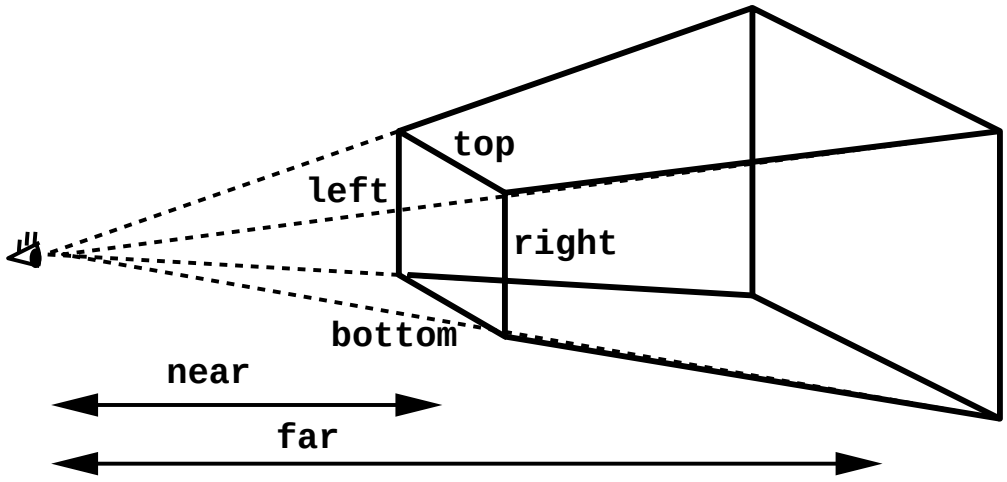
ビューイング・ボリュームの形状は直方体
その特徴は次の通り

- 遠くのものと近くのものが同じ大きさのまま変換される
  遠近感がなく3次元的奥行きに欠ける平面的な画像
- 投影面に平行な物の距離が正確に計れる
- ワールド空間のZ値から正規化座標系のZ値にリニアにマッピングされる
  透視投影の場合に比べビューイング・ボリュームのnear面とfar面の設定は神経質にならないで良い
- 平行投影変換の設定関数は次の2種類がある
  - glOrtho
  - gluOrtho2D

[1] 透視投影変換の場合、ワールド空間のZ値から正規化座標系のZ値へのマッピングが非線形であるため、nearとfarを広げると遠近の解像度が極端に悪くなる場合がある

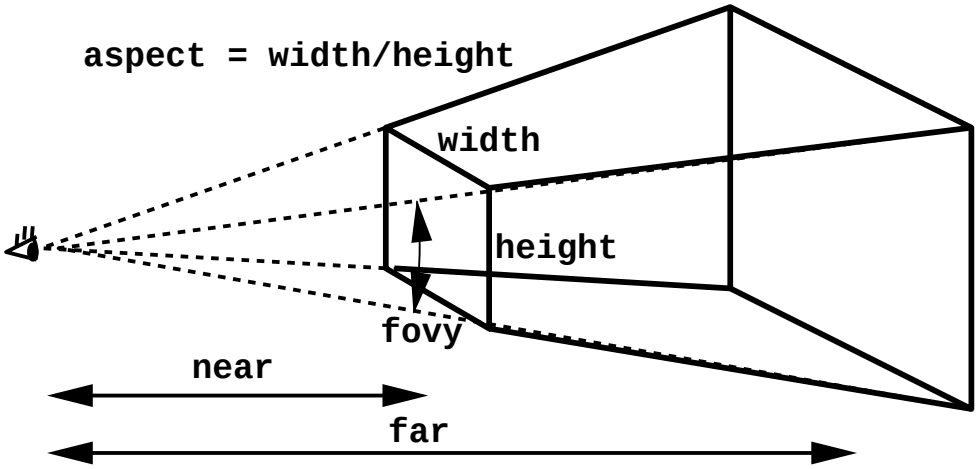
透視投影変換の設定関数(1)
– glFrustum –

- void glFrustum (GLdouble left, GLdouble right,
  GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
- 視線方向に対し上下、左右が非対象のビューイング・ボリュームを設定できる
  自由度が高い反面、引数が多く使い難い場合がある
- frustum は錐台、切頭体という意味



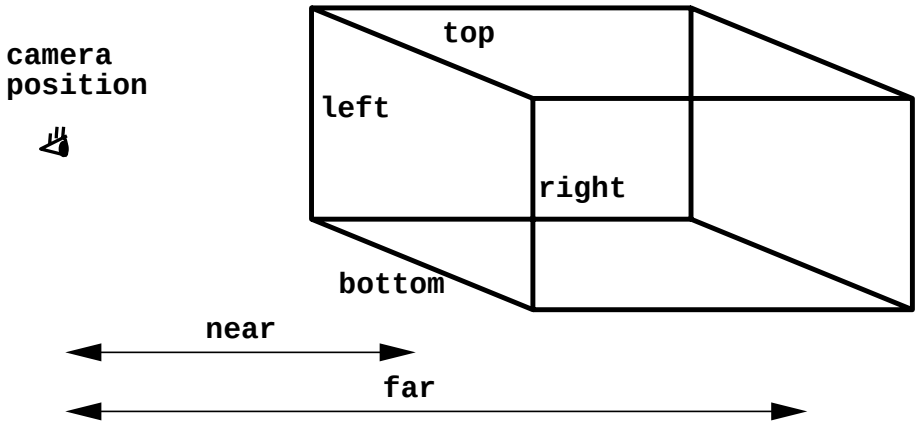
透視投影変換の設定関数(2)
– gluPerspective –

- void gluPerspective (GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar);
- 視線方向に対し上下、左右が対象のビューイング・ボリュームを設定できる
- 引数が少なく設定が簡単



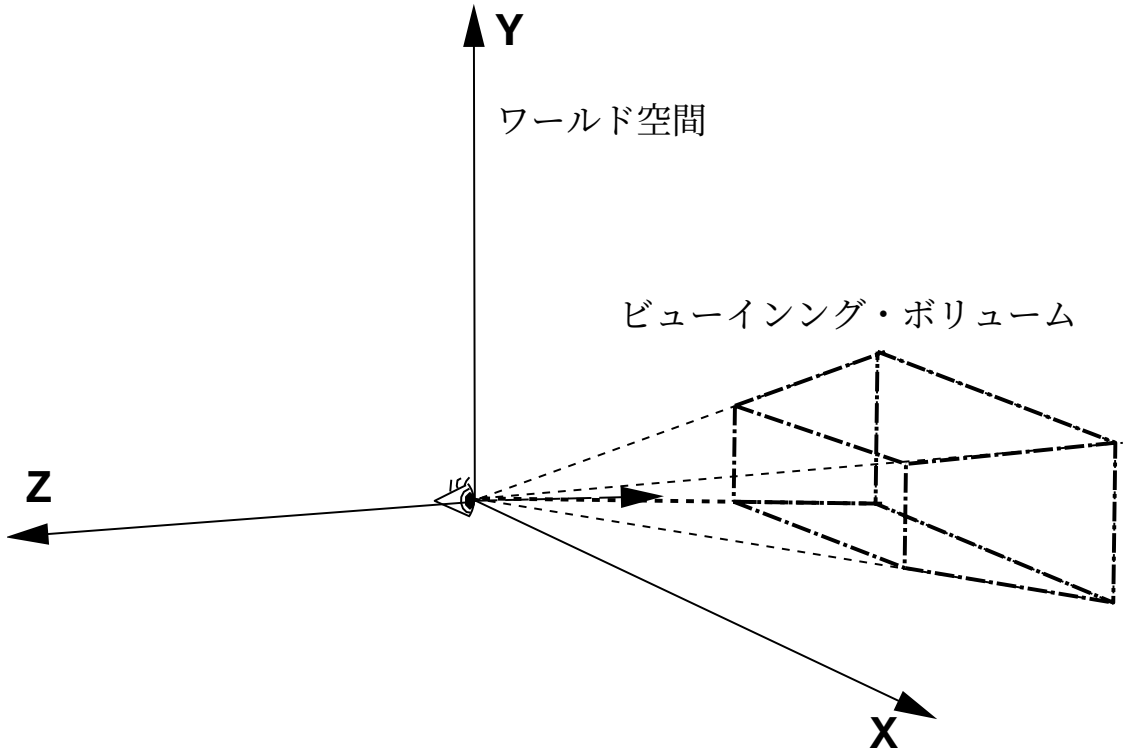
平行投影変換の設定関数 glOrtho サブルーチン

- void glOrtho (GLdouble left, GLdouble right,
  GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
- void gluOrtho2D (GLdouble left, GLdouble right,
  GLdouble bottom, GLdouble top);



投影変換とビューイング・ボリュームの位置

- 投影変換設定後、ワールド空間の原点を視点位置にしてビューイング・ボリュームが設定されている



```
 1 /*
 2 ** projection.c - 投影変換のみの３次元グラフィックス
 3 */
 4
 5 #include <GL/glut.h>
 6 #include "laconia.h" /* セミナー用３次元基本形状 */
 7
 8 void
 9 initialize(void)
10 {
11   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
12 }
13
14 void
15 display(void)
16 {
17   glClear( GL_COLOR_BUFFER_BIT );
18
19   glColor3f( 1.0f, 1.0f, 0.0f );
20
21   /* 球の描画 */
22   lacSphere( 10.0, 'w' );
23
24   glutSwapBuffers();
25 }
26
27 void
28 keyboard( unsigned char c, int x, int y )
29 {
30   if( c==27 ) exit( 0 );
31 }
32
33 void
34 reshape( int w, int h )
35 {
36   glViewport( 0, 0, w, h );
37
38   /* ３次元投影マトリックスの設定 */
39   glMatrixMode( GL_PROJECTION );
40   glLoadIdentity();
41   gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
42   glMatrixMode( GL_MODELVIEW );
43
```

```
 44 }
 45
 46 void
 47 animate(void)
 48 {
 49   /* 演習で、毎ループ reshape 関数を使って投影変換のパラメータ
が、
 50      更新されるようにするため、ここで reshape を呼んでいる。
 51      演習のとき以外は reshape を呼ぶ必要はない。 */
 52   reshape(glutGet(GLUT_WINDOW_WIDTH), glutGet(GLUT_WINDOW_HEIGHT));
 53   glutPostRedisplay();
 54 }
 55
 56 int
 57 main( int argc, char *argv[] )
 58 {
 59   glutInit( &argc, argv );
 60
 61   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE );
 62   glutInitWindowPosition( 100, 100 );
 63   glutInitWindowSize( 500, 400 );
 64   glutCreateWindow( argv[0] );
 65
 66   glutDisplayFunc( display );
 67   glutKeyboardFunc( keyboard );
 68   glutReshapeFunc( reshape );
 69   glutIdleFunc( animate );
 70
 71   initialize();
 72   glutMainLoop();
 73   return 0;
 74 }
 75
```

<演習>

1. forYou ディレクトリの projection プログラムを実行して下さい。ただし、ここでは原点を中心とする球を、原点から見るので、球全体を見ることはできません。

```
% cd forYou
% make projection
% projection
```

2. 作業ディレクトリの animation プログラムにワイヤーフレームの 3 次元形状を付加し、3 次元投影変換を組み込んで下さい。

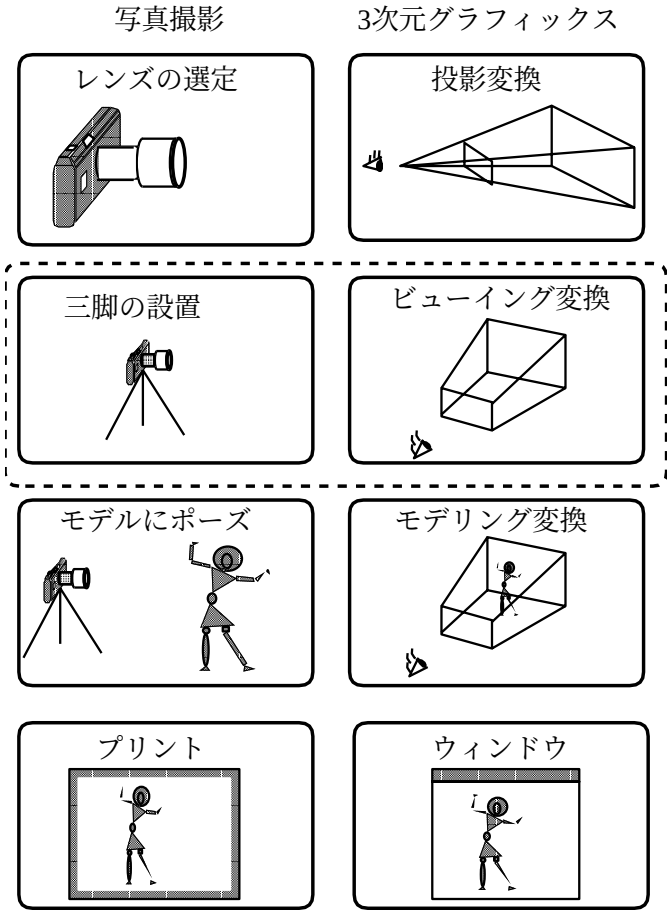
```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

3. さらに animation プログラムの 3 次元投影変換のパラメーターが変化するようにして下さい。

例:
```
draw(){
    ...
    static float fov = 10.0;
    ...
    fov += 1.0;
    if(fov > 120.0)
    {
      fov = 10.0;
    }
    ...
    gluPerspective(fov, 1.0, 0.1, 200.0);
    ...
    }
```

3.3　ビューイング変換の設定

ビューイング変換と写真撮影の関係



- ビューイング変換はビューイング・ボリュームをワールド空間内に配置する
- ビューイング変換の設定はカメラをどの位置に置きどの方向に向けるかということに相当する

ビューイング変換設定の流れ

- 投影変換の後に設定する[2]
- ビューイング変換の設定は3つのステップに分かれる
  1. 処理対象マトリックスをビューイング変換を担当するModelviewマトリックスにする
  2. Modelviewマトリックスを単位行列で初期化する
  3. gluLookAtなどのビューイング変換設定関数でカメラの位置と方向を指定する

[2] 投影変換前に設定することも可能だが、マトリックス・モードの切替回数を最小化するという観点から投影変換の後に設定する方がよい

ビューイング変換設定関数

- GLU ライブラリに gluLookAt 一種類が用意されている
- 観察者も観察点も移動する場合に適したビューイング変換
  gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz);[3]

  eyex, eyey, eyez 観察者の目の位置、ビュー・ポイント

  centerx, centery, centerz リファレンス・ポイント、視線上の点

  upx, upy, upz 頭の天辺の向く方向
- ビューイング変換はモデリング変換 (後出) の組み合わせで実装できる

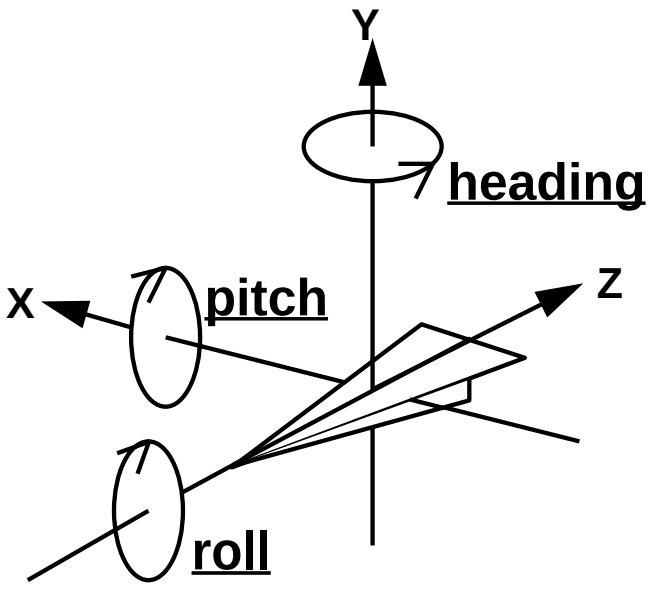
[3] IRIS GL の lookat でなく、4Dgifts に含まれていた upat 関数に相当する

フライト・シミュレーション用のビューイング変換

GLU でサポートしている gluLookAt 関数の他にもモデリング変換を組み合わせると、便利なビューイング変換ができる

- GLU でサポートしている gluLookAt 関数の他にもモデリング変換を組み合わせると、便利なビューイング変換ができる
- フライトシミュレータ向きのビューイング変換

```
void pilotView(float x, float y, float z,
               float head, float pitch, float roll)
{
    glRotatef(roll,    0, 0, 1);
    glRotatef(-pitch,   1, 0, 0);
    glRotatef(-heading, 0, 1, 0);
    glTranslatef(-x,  -y,  -z);
}
```



CAD アプリケーション用のビューイング変換

- CAD など原点近傍のモデルを観察するのに便利なビューイング変換
- polarView(dist, azim, inc, twist);

dist 目からワールド空間に於ける原点までの距離

azim xy 平面上の y 軸からの方位角

inc　yz 平面上の z 軸からの入射角

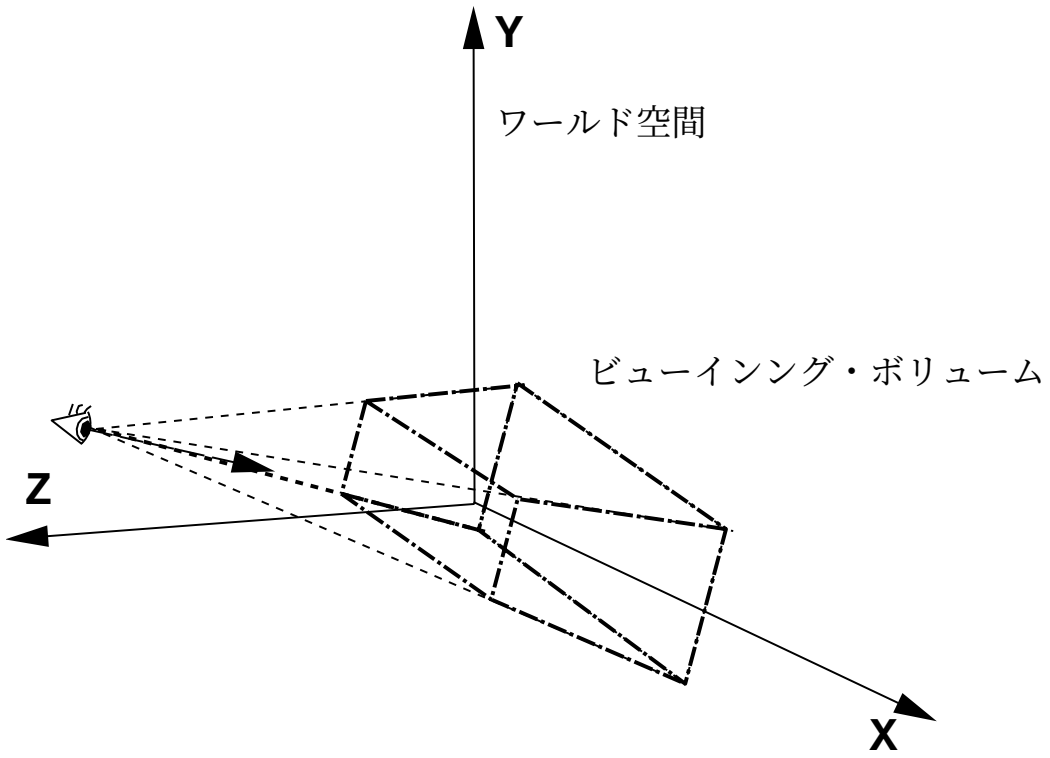
twist 視線を軸とする回転量

- IRIS GL では関数として提供されていたが OpenGL では以下のコードで代用する

```
void polarView{GLdouble distance, GLdouble twist,
   GLdouble elevation, GLdouble azimuth)
{
    glTranslated(0.0, 0.0, -distance);
    glRotated(-twist, 0.0, 0.0, 1.0);
    glRotated(-elevation, 1.0, 0.0, 0.0);
    glRotated(azimuth, 0.0, 0.0, 1.0);
}
```

ビューイング変換とビューイング・ボリュームの位置

- ビューイング変換により、ビューイング・ボリュームはワールド空間中の指定した位置に設置される



```
 1 /*
 2 ** viewing.c - 投影変換とビューイング変換
 3 */
 4
 5 #include <GL/glut.h>
 6 #include "laconia.h"
 7
 8 void
 9 initialize(void)
10 {
11   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
12 }
13
14 void
15 display(void)
16 {
17   /* ビューイングマトリックスの設定 */
18   /* 演習で、描画の度にビューイング変換パラメータを
19      更新するため、ここで設定する
20      視点位置不動なら initialize 関数内で設定可能 */
21   glLoadIdentity();
22   gluLookAt( 0.0, 50.0, 100.0,
23              0.0,  0.0,   0.0,
24              0.0,  1.0,   0.0 );
25
26   glClear( GL_COLOR_BUFFER_BIT );
27
28   glColor3f( 1.0f, 1.0f, 0.0f );
29   lacSphere( 10.0, 'w' );
30
31   glutSwapBuffers();
32 }
33
34 void
35 keyboard( unsigned char c, int x, int y )
36 {
37   if( c==27 ) exit( 0 );
38 }
39
40 void
41 reshape( int w, int h )
42 {
43   glViewport( 0, 0, w, h );
```

```
44
45   glMatrixMode( GL_PROJECTION );
46   glLoadIdentity();
47   gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
48   glMatrixMode( GL_MODELVIEW );
49 }
50
51 void
52 animate(void)
53 {
54   glutPostRedisplay();
55 }
56
57 int
58 main( int argc, char *argv[] )
59 {
60   glutInit( &argc, argv );
61
62   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE );
63   glutInitWindowPosition( 100, 100 );
64   glutInitWindowSize( 500, 400 );
65   glutCreateWindow( argv[0] );
66
67   glutDisplayFunc( display );
68   glutKeyboardFunc( keyboard );
69   glutReshapeFunc( reshape );
70   glutIdleFunc( animate );
71
72   initialize();
73   glutMainLoop();
74   return 0;
75 }
76
```

<演習>

1. forYou ディレクトリの viewing プログラムを実行して下さい。

```
% cd forYou
% make viewing
% viewing
```

2. 作業ディレクトリの animation プログラムを、3次元投影変換が変化しないようにして下さい。その後、ビューイング変換を加え、各モデルを適切に観察できるように視点を設定して下さい。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

3. そして今度はビューイング変換のパラメーターが変化するようにして下さい。

例:
```
draw(){
...
static float eyeX = -100.0;
...
eyeX += 1.0;
if(eyeX > 100.0)
{
  eyeX = -100.0;
}
...
gluLookAt(eyeX, -100.0, 100.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
...
}
```

サブルーチンのまとめ

投影変換の指定

```
void gluPerspective(GLdouble fovy, GLdouble aspect,
                    GLdouble near, GLdouble far);

void glFrustum(GLdouble left, GLdouble right,
               GLdouble bottom, GLdouble top,
               GLdouble near, GLdouble far);

void glOrtho(GLdouble left, GLdouble right,
             GLdouble bottom, GLdouble top,
             GLdouble near, GLdouble far);

void gluOrtho2(GLdouble left, GLdouble right,
               GLdouble bottom, GLdouble top);
```

ビューイング変換の指定

```
void glLookAt(GLdouble vx, GLdouble vy, GLdouble vz,
              GLdouble px, GLdouble py, GLdouble pz,
              GLdouble tx, GLdouble ty, GLdouble tz);

void polarView(GLdouble dist, GLdouble azim,
               GLdouble inc, GLdouble twist); /* 非標準 */

void pilotView(GLdouble planex, GLdouble planey,
              GLdouble planez, GLdouble roll,
              GLdouble pitch, GLdouble heading);/* 非標準 */
```

第4章　3次元グラフィックスの応用

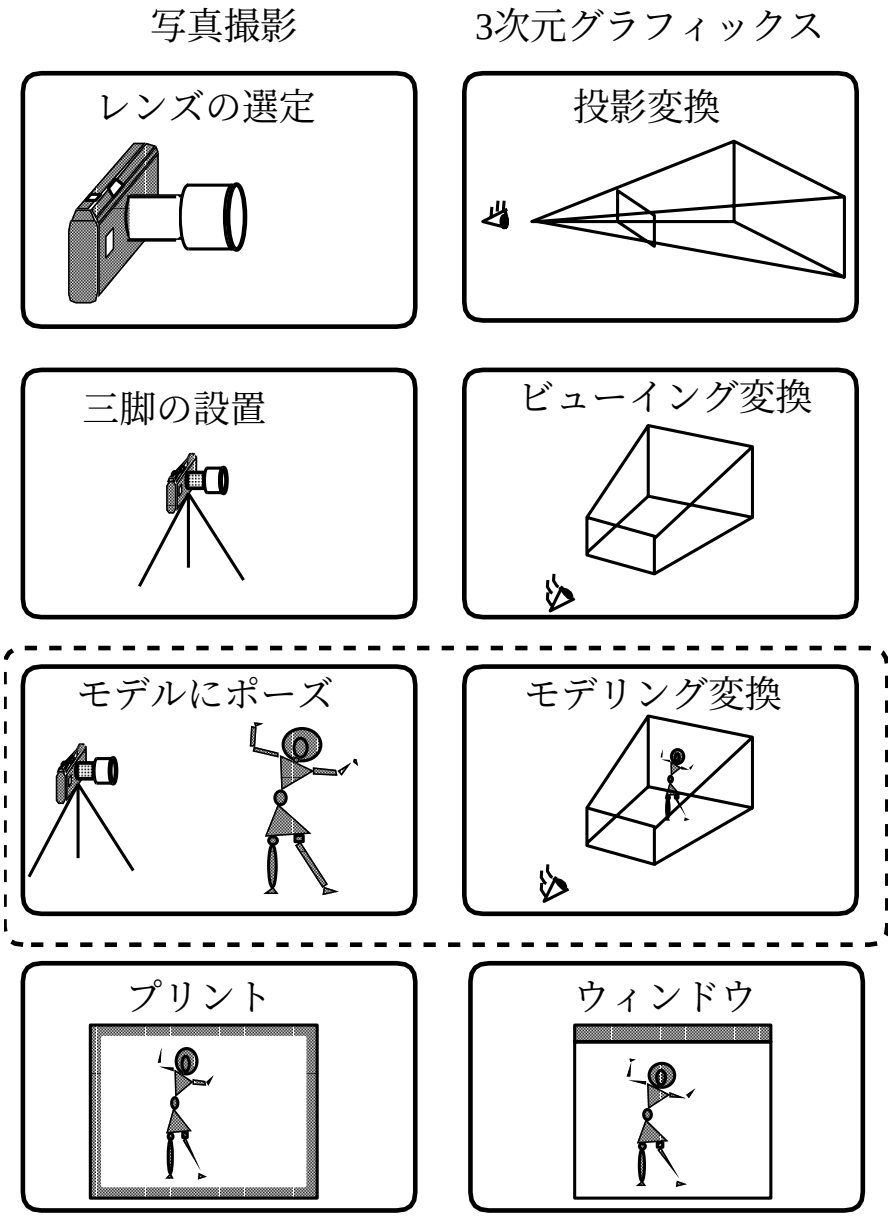
本章の目的

- 各種モデリング変換の使用法
- 各種座標系の関係について理解する
- モデリング変換を使った3次元プログラムの作成
- 変換マトリックスの仕組とスタックの利用
- モデリング変換の合成

4.1　モデリング変換

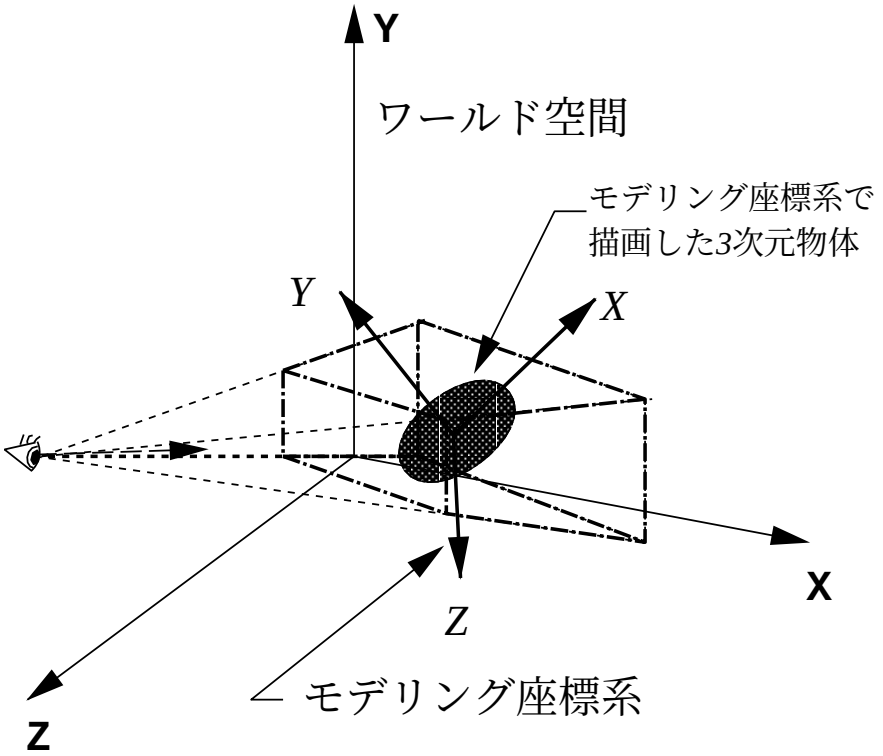
モデリング変換と写真撮影の関係

- モデリング変換により物体を 3 次元ワールド空間の任意の位置に任意の大きさする



モデリング変換とモデリング座標系

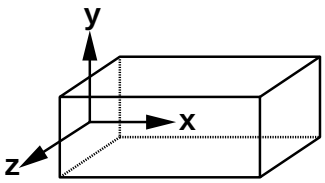
- 物体を描くためのローカルな座標系をモデリング座標系という
- モデリング座標系の原点はモデリング変換を施さない状態ではワールド空間の原点に一致している
- モデリング座標系はモデリング変換を使いワールド空間中の任意の位置に配置したり、拡大縮小したり、回転したりできる

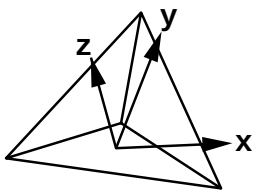


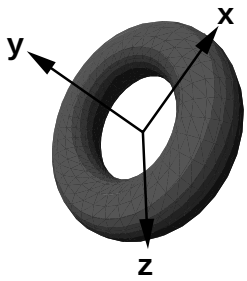
- glBegin などの描画関数はモデリング座標系をベースに 3 次元物体の描画を行う

モデリング変換の性質

- モデリング変換では平行移動、回転移動、拡大縮小ができる
- 平行移動、回転移動、拡大縮小したモデリング座標上で物体を描画することにより、物体の移動、回転、拡大縮小をコントロールできる







- モデリング変換の関数はカレントの変換マトリックスに対して掛け合わせを行う
  Matrix モードをモデルビュー・マトリックスにしておく
- モデリング変換とビューイング変換は任意の 3 点間の角度が保存される
  – 光源モデルの計算のときに重要な性質で、アフィン変換の一種である
  – アフィン変換は 2 直線の平行が保存される変換をいう

モデリング変換の設定関数

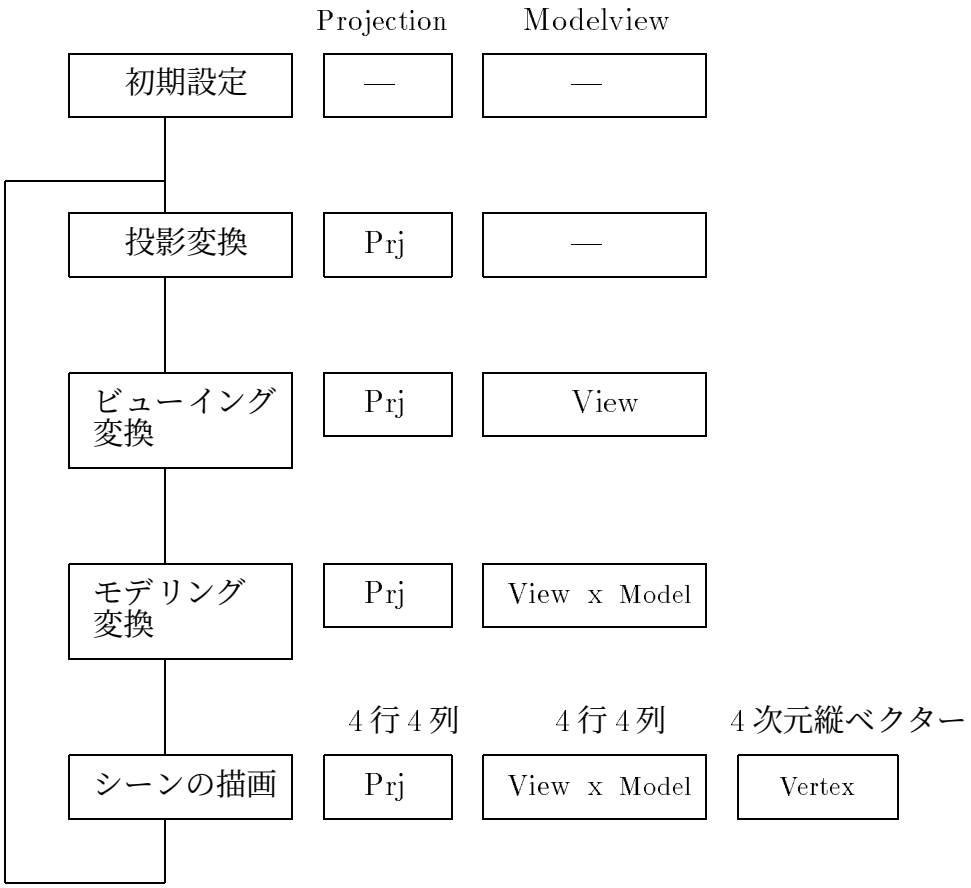
- カレントの操作対象マトリックスを Modelview マトリックスにする
  （すでに Modelview マトリックスが操作対象になっていれば設定不要）

```
glMatrixMode(GL_MODELVIEW);
```

- 回転変換
  - glRotated(GLdouble angle, GLdouble x, GLdouble y, GLdouble z);
  - glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z);
  - angle は度で指定する（radian ではないので注意）
  - x, y, z は回転の中心となるベクターのそれぞれの成分
- スケーリング変換
  - glScaled(GLdouble x, GLdouble y, GLdouble z);
  - glScalef(GLfloat x, GLfloat y, GLfloat z);
  - x、y、z は各軸方向への拡大率
  - 負を指定して鏡対称な図形を描画することも可能
  - スケーリング変換は原点からの距離に依存する
- 平行移動
  - glTranslated(GLdouble x, GLdouble y, GLdouble z);
  - glTranslatef(GLfloat x, GLfloat y, GLfloat z);
  - x、y、z は各軸方向への移動量

3次元変換の流れとカレント・マトリックスの推移

- 投影変換からモデリング変換までの変換マトリックスの推移を確認する



OpenGLにおけるマトリックス演算の方法

- 3次元物体を構成する頂点はx、y、zの3要素に1を4番目の要素とする4次元の縦ベクターで表現する
- 変換マトリックスは4行4列のものを使い縦方向要素が連続するように1次元配列に保存される
- 4次元ベクターと4行4列のマトリックスを使うのは透視投影変換、並行移動をマトリックスとベクターの演算で取り扱うためである
  この4次元空間は同次座標系 (Homogeneous Coordinates) という
- 演算の結果、4番目の要素が1でない数字になった場合は各要素を4番目の要素wで割ると3次元の座標に戻すことができる

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

モデリング変換のマトリックス (1)
– 移動とスケーリング –

- 平行移動のマトリックスは以下の通り

$$M_{Translate} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- スケーリング変換を行なうマトリックスは以下の通り

$$M_{Scale} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

モデリング変換のマトリックス (2)
– 回転 –

X 軸、Y 軸、Z 軸を中心に回転を行なうマトリックスはそれぞれ次のようになっています。

$$glRotate(\theta,1,0,0) : M_{RotX}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$glRotate(\theta,0,1,0) : M_{RotY}(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$glRotate(\theta,0,0,1) : M_{RotZ}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
 1 /*
 2 ** modeling1.c - 単純なモデリング変換
 3 */
 4
 5 #include <GL/glut.h>
 6 #include "laconia.h"
 7
 8 void
 9 initialize(void)
10 {
11   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
12 }
13
14 void
15 display(void)
16 {
17   /* モデリング変換のパラメータ */
18   static float offsetX = -51.0;
19
20   /* パラメータの計算 (offsetX: -50.0~50.0) */
21   offsetX += 1.0;
22   if( offsetX >= 51.0 ) offsetX -= 101.0;
23
24   glClear( GL_COLOR_BUFFER_BIT );
25
26   /* 視点位置の設定 */
27   glMatrixMode( GL_MODELVIEW );
28   glLoadIdentity();
29   gluLookAt( 0.0, 50.0, 100.0,
30              0.0,  0.0,   0.0,
31              0.0,  1.0,   0.0 );
32
33   /* モデリングマトリックスの設定 */
34   glTranslatef( offsetX, 0.0, 0.0 );
35
36   glColor3f( 1.0f, 1.0f, 0.0f );
37   lacSphere( 10.0, 'w' );
38
39   glutSwapBuffers();
40 }
41
42 void
43 keyboard( unsigned char c, int x, int y )
```

```
44 {
45   if( c==27 ) exit( 0 );
46 }
47
48 void
49 reshape( int w, int h )
50 {
51   glViewport( 0, 0, w, h );
52
53   glMatrixMode( GL_PROJECTION );
54   glLoadIdentity();
55   gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
56   glMatrixMode( GL_MODELVIEW );
57 }
58
59 void
60 animate(void)
61 {
62   glutPostRedisplay();
63 }
64
65 int
66 main( int argc, char *argv[] )
67 {
68   glutInit( &argc, argv );
69
70   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE );
71   glutInitWindowPosition( 100, 100 );
72   glutInitWindowSize( 500, 400 );
73   glutCreateWindow( argv[0] );
74
75   glutDisplayFunc( display );
76   glutKeyboardFunc( keyboard );
77   glutReshapeFunc( reshape );
78   glutIdleFunc( animate );
79
80   initialize();
81   glutMainLoop();
82   return 0;
83 }
84
```

<演習>

1. forYou ディレクトリの modeling1 プログラムを実行して下さい。

```
% cd forYou
% make modeling1
% modeling1
```

2. 作業ディレクトリの animation プログラムにモデリング変換を加えて下さい。モデリング変換はビューイング変換の直後に行ない、そのとき全てのモデルが移動します。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

3. さらにモデリング変換のパラメーターが変化するようにして下さい。

4.2 マトリックス・スタックとモデリング変換の合成

マトリックス・スタックの用途

- マトリックス・スタックを利用すると変換マトリックスを保存（プッシュ）しておいて、後から取り出す（ポップ）ことができる
- マトリックス・スタックを使えば複数の物体に独立したモデリング変換を作用させることができる
- マトリックス・スタックを利用すると投影変換やビューイング変換の設定回数を減らし、各種変換設定の合理化が図れる
- マトリックス・スタックを使うと 3 次元物体と 2 次元テキストを描き分けることができる

マトリックス・スタックの使い方

- OpenGL には Projection、Modelview、Texture の 3 種類の変換マトリックスがあり各々にマトリックス・スタックが用意されいる
- 各マトリックス用のスタックは次の段数用意されている

  `Modelview` マトリックス ： 32 段
  `Projection` マトリックス ： 2 段以上
  `Texture` マトリックス ： 2 段以上

- マトリックスの保存、および取り出しは glMatrixMode 関数で設定されているカレントの操作対象マトリックスに対して行われる
- カレントの変換マトリックスのコピーをマトリックス・スタックに保存する
  void glPushMatrix();
- マトリックス・スタックの最上位から変換マトリックスを取り出して、カレントの変換マトリックスを置き換える
  void glPopMatrix();
- その他のマトリックス操作関数
  次の関数は、glMatrixMode 関数の影響を受ける
  – カレントの変換マトリックスを m で置き換える
    void glLoadMatrix(Matrix m);
  – カレントの変換マトリックスに m を掛け合わせる
    void glMultMatrix(Matrix m);

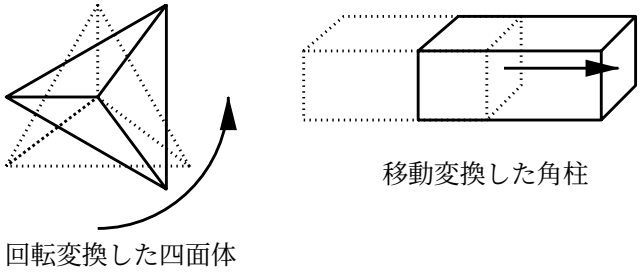
マトリックス情報の問い合わせ

- 次の 2 つの関数を使い、各種マトリックス関連情報の問い合わせができる
  void glGetFloatv(GLenum pname, GLfloat *m);
  void glGetDoublev(GLenum pname, GLdouble *m);
  void glGetIntegerv(GLenum pname, GLint *params);
- これらは glMatrixMode 関数の影響を受けない
- 第一引数の pname に以下の定数を指定することにより各種マトリックス情報を取り出せる
  - カレントの変換マトリックスを配列 m に書き込む
    GL_PROJECTION_MATRIX、GL_MODELVIEW_MATRIX、GL_TEXTURE_MATRIX
  - カレントのマトリックス・モードを調べる
    GL_MATRIX_MODE
  - カレントのマトリックス・スタックの深さを調べる
    GL_PROJECTION_STACK_DEPTH、GL_MODELVIEW_STACK_DEPTH、
    GL_TEXTURE_STACK_DEPTH
  - マトリックス・スタックの深さの上限を調べる
    GL_MAX_MODELVIEW_STACK_DEPTH、GL_MAX_PROJECTION_STACK_DEPTH、
    GL_MAX_TEXTURE_STACK_DEPTH

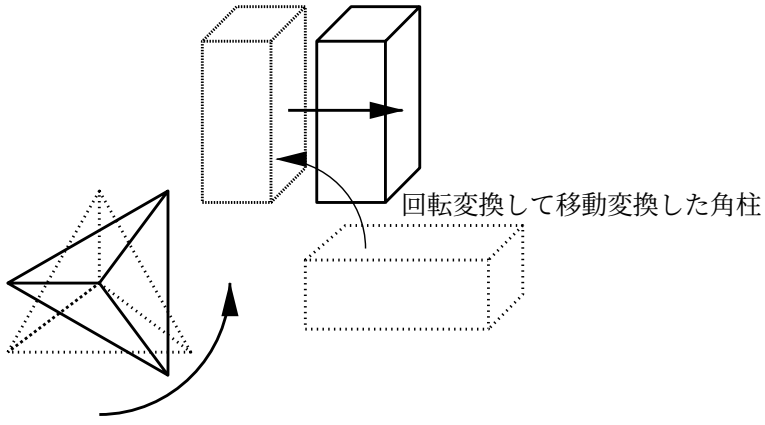
マトリックス・スタックの応用 (1)
– 独立したモデリング変換 –

- 次の (a) ように回転のモデリング変換だけが作用した四面体と、移動のモデリング変換だけが作用した角柱を描く場合を考える

- 単純にモデリング変換を繰り返すと、(b) のように回転のモデリング変換が作用した四面体と回転と移動のモデリング変換が作用した角柱が描画される

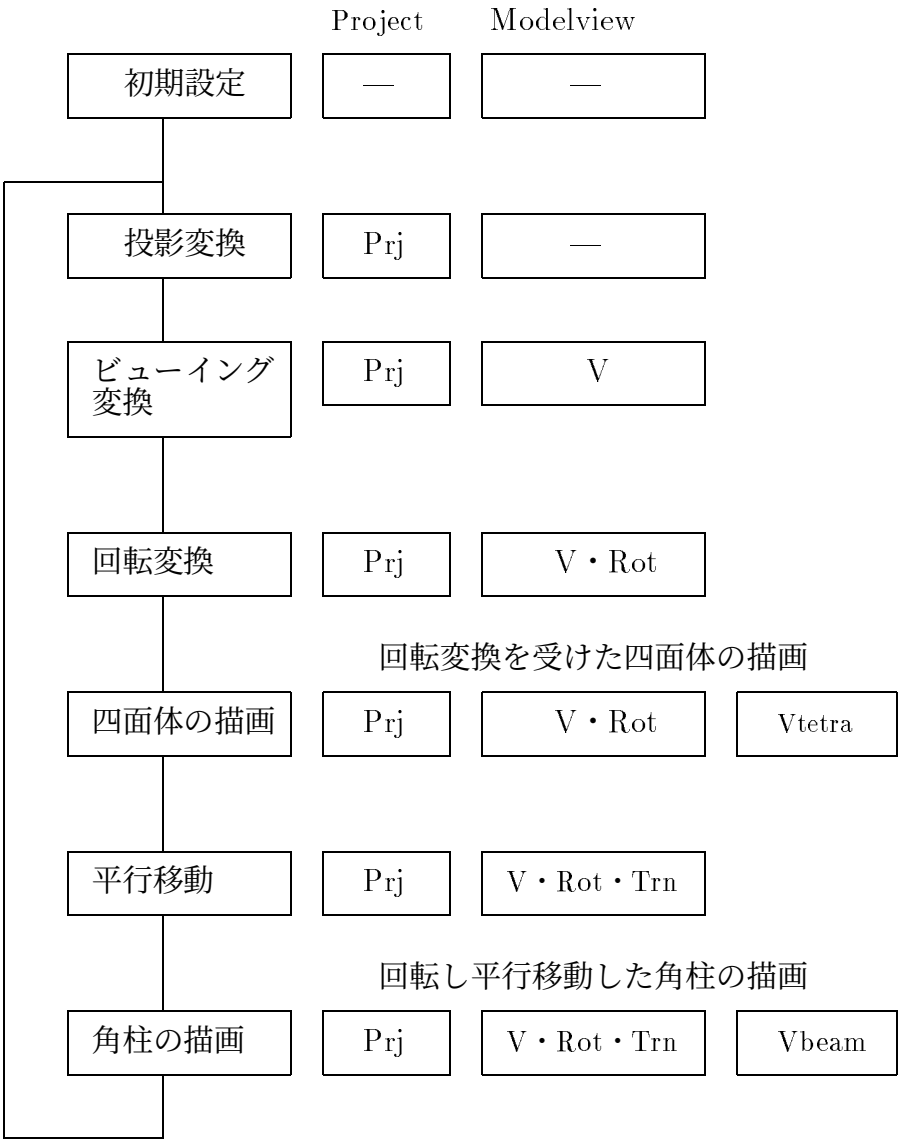


(a) 独立したモデリング変換の例 (本節の目的)



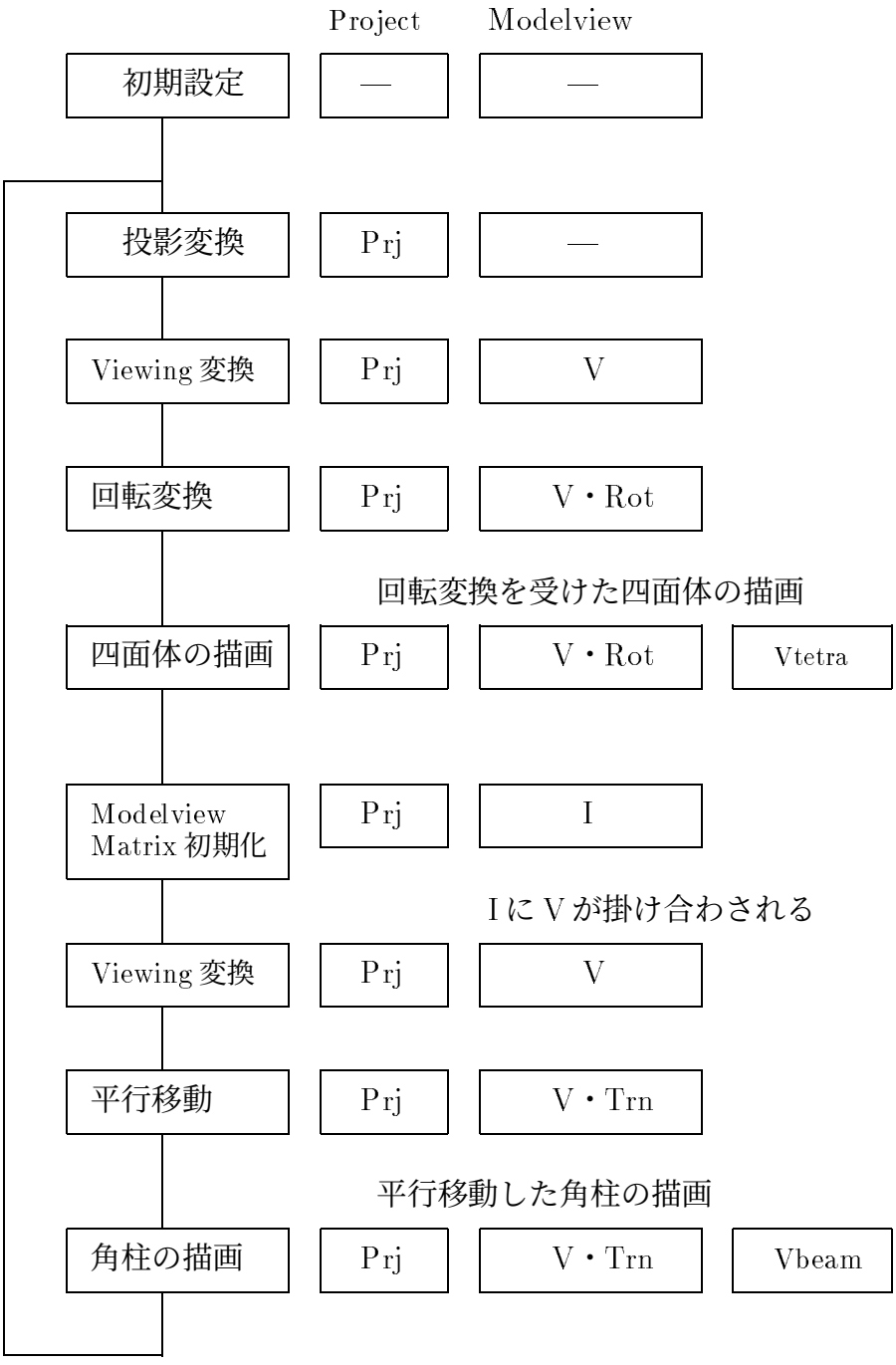
(b) 独立していないモデリング変換の例 (簡単)

独立していないモデリング変換の流れ図



- この流れ図では角柱に移動と回転の両方のモデリング変換が作用し狙った絵が得られない。

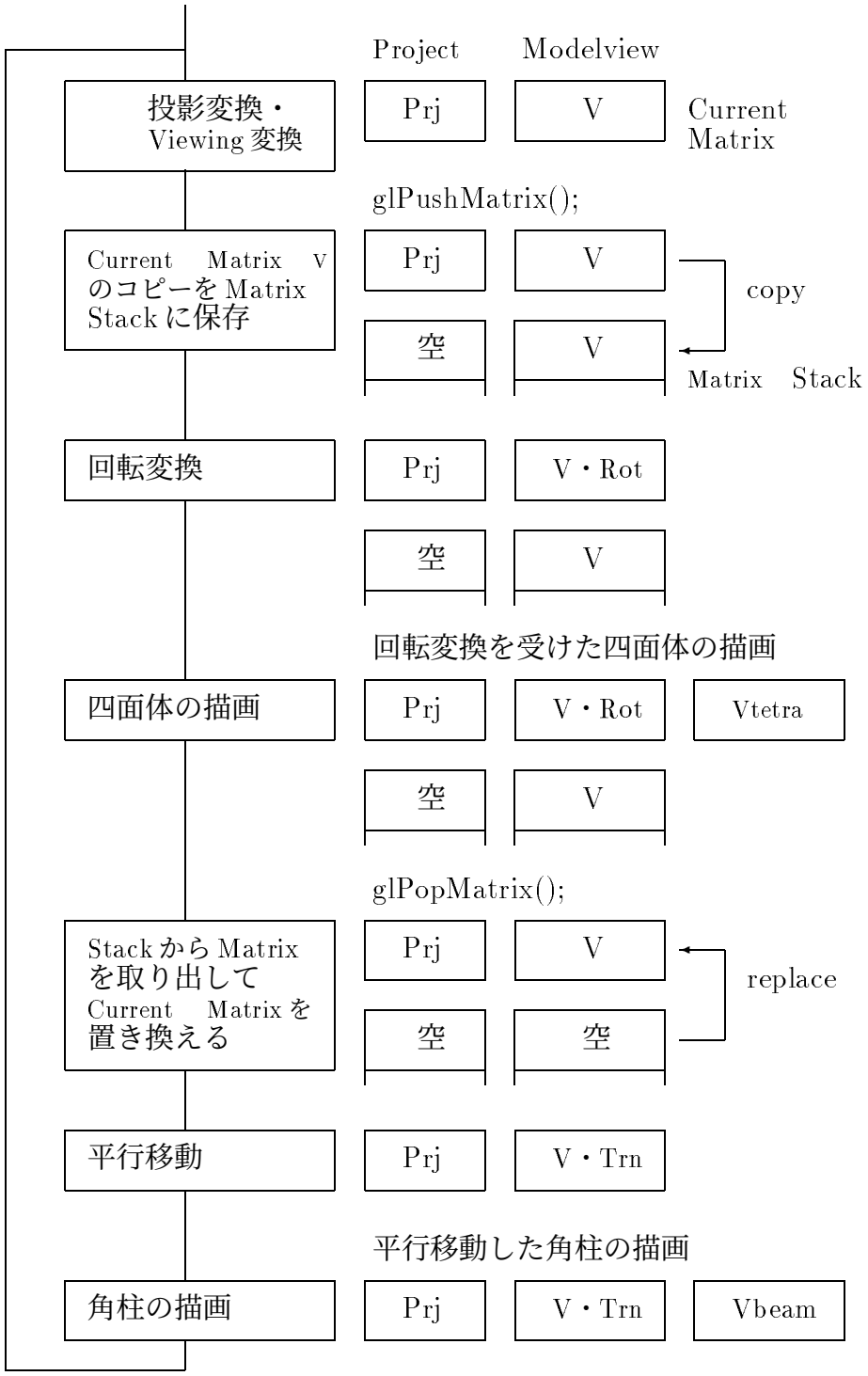
独立したモデリング変換の流れ図 – 無駄な例 –



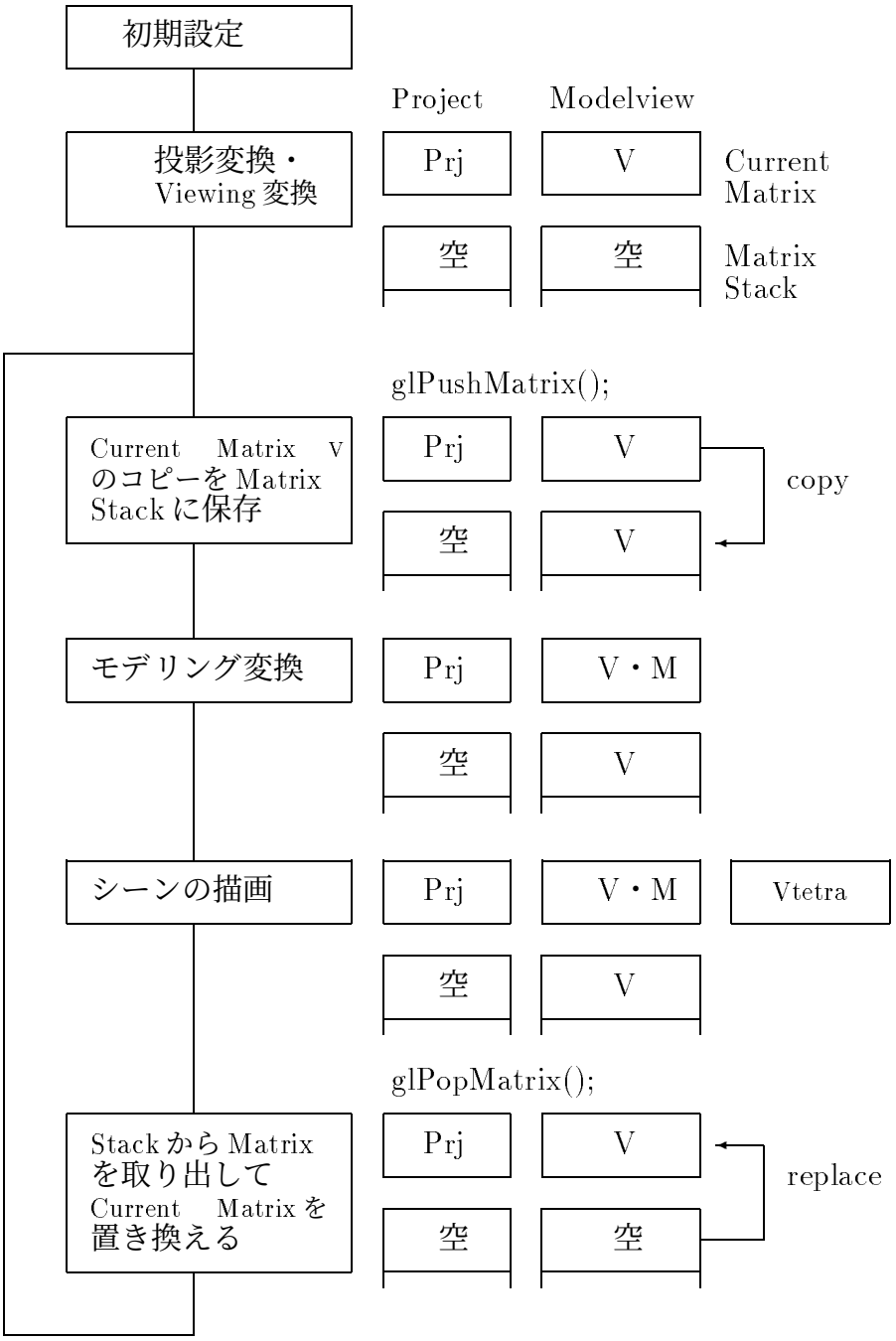
- 目的の絵は得られるが Modelview マトリックスの初期化とビューイングを 2 度繰り返し無駄がある

独立したモデリング変換の流れ図
– スタックを利用 –

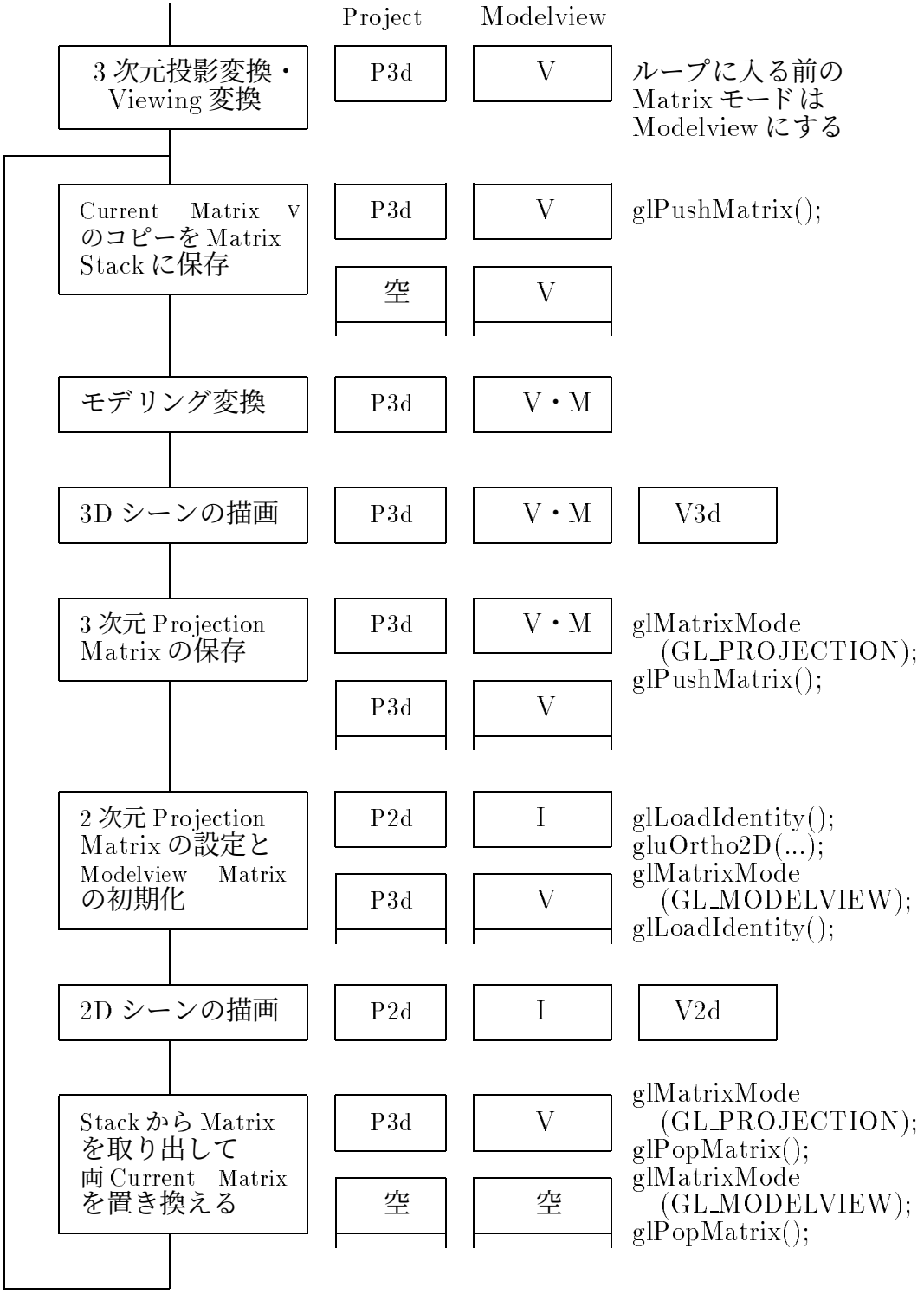
必要なマトリックスをスタックに保存して再利用する



マトリックス・スタックの応用 (2)
– ビューイング変換の最適化 –



マトリックス・スタックの応用(3)
– 2次元と3次元の混合プログラミング –



```
 1 /*
 2 ** modeling2.c - 独立した動きをする 2 つのモデル
 3 */
 4
 5 #include <GL/glut.h>
 6 #include "laconia.h"
 7
 8 void
 9 initialize(void)
10 {
11   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
12
13   /* 視点位置が固定のため初期化時に一回設定する */
14   glMatrixMode( GL_MODELVIEW );
15   glLoadIdentity();
16   gluLookAt( 0.0, 50.0, 100.0,
17              0.0,  0.0,   0.0,
18              0.0,  1.0,   0.0 );
19 }
20
21 void
22 display(void)
23 {
24   /* モデリング変換のパラメータ */
25   static float offsetX = -51.0;
26   static float angle = -1.0;
27
28   /* offsetX の計算 (-50.0~50.0) */
29   offsetX += 1.0;
30   if( offsetX>=51.0 ) offsetX -= 101.0;
31
32   /* angle の計算 (0.0~359.0) */
33   angle += 1.0;
34   if( angle>=360.0 ) angle -= 360.0;
35
36   glClear( GL_COLOR_BUFFER_BIT );
37
38   /* 平行移動する球 */
39   glPushMatrix();
40   glTranslatef( offsetX, 0.0, 0.0 );
41   glColor3f( 1.0f, 1.0f, 0.0f );
42   lacSphere( 10.0, 'w' );
43   glPopMatrix();
```

```
44
45   /* 回転する円錐 */
46   glPushMatrix();
47   glRotatef( angle, 0.0, 1.0, 0.0 );
48   glColor3f( 0.0f, 0.0f, 1.0f );
49   lacCone( 15.0, 15.0, 'w' );
50   glPopMatrix();
51
52   glutSwapBuffers();
53 }
54
55 void
56 keyboard( unsigned char c, int x, int y )
57 {
58   if( c==27 ) exit( 0 );
59 }
60
61 void
62 reshape( int w, int h )
63 {
64   glViewport( 0, 0, w, h );
65
66   glMatrixMode( GL_PROJECTION );
67   glLoadIdentity();
68   gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
69   glMatrixMode( GL_MODELVIEW );
70 }
71
72 void
73 animate(void)
74 {
75   glutPostRedisplay();
76 }
77
78 int
79 main( int argc, char *argv[] )
80 {
81   glutInit( &argc, argv );
82
83   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE );
84   glutInitWindowPosition( 100, 100 );
85   glutInitWindowSize( 500, 400 );
86   glutCreateWindow( argv[0] );
87
```

```
 88   glutDisplayFunc( display );
 89   glutKeyboardFunc( keyboard );
 90   glutReshapeFunc( reshape );
 91   glutIdleFunc( animate );
 92
 93   initialize();
 94   glutMainLoop();
 95   return 0;
 96 }
 97
```

<演習>

1. forYou ディレクトリの modeling2 プログラムを実行して下さい。

```
% cd forYou
% make modeling2
% modeling2
```

2. animation プログラムの各モデルに、独立した動きを与えて下さい。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

4.3　モデリング変換の合成

モデリング変換の合成

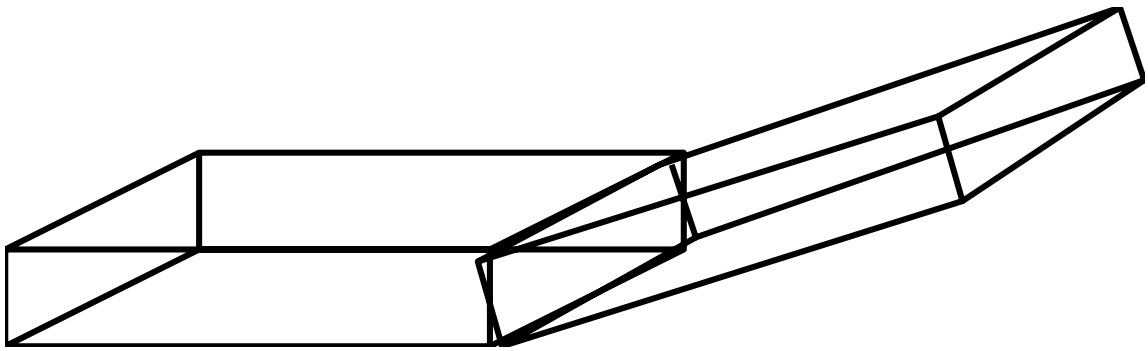
- モデリング変換の合成はモデリング変換の設定関数を複数回呼ぶ
- モデリング変換の設定関数を呼ぶたびにカレントの変換マトリックスに対しモデリング変換のマトリックスが掛け合わされる
- 再利用したい変換マトリックスは glPushMatrix でスタックに保存し glPopMatrix で取り出して利用する
- モデリング変換の合成はローカルなモデリング座標系を想定し、それに対し順次変換を施していくと考え易い
- モデリング座標系の原点はモデリング変換を施す前はワールド空間の原点に一致している
- ローカルなモデリング座標系の移動でモデリング変換の合成を考える

- モデリング変換の合成は次のようなときに使う
  – 任意の点を中心とした拡大や、任意の軸を中心とする回転
  – ロボットの腕のように関節を持つ階層構造のあるオブジェクトを動かすとき

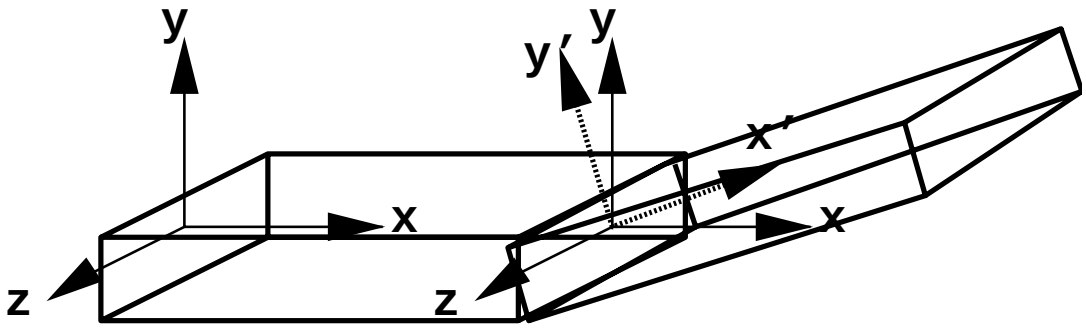
- モデリング座標系は右手座標系
- ワールド空間中に貼りついている
- 3 次元モデル形状を定義するのに使う

モデリング変換の合成(1)
– ロボット・アーム –

- モデリング変換を使って次のようなロボット・アームを自由にコントロールするプログラムを作成する

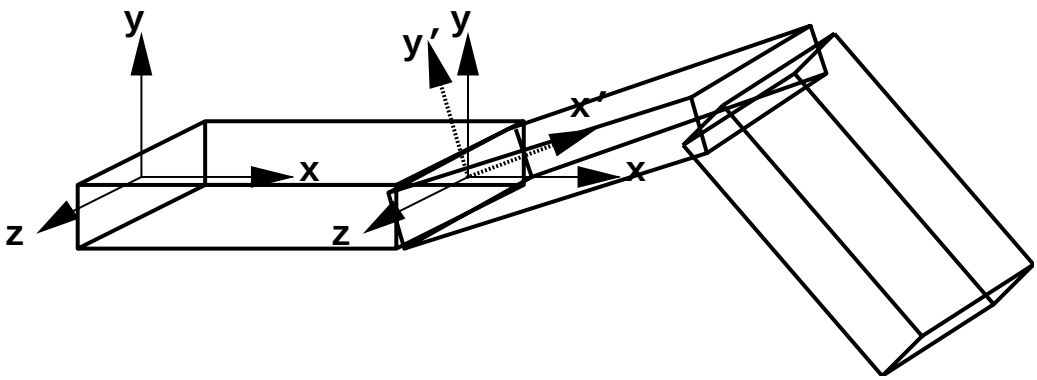


ロボット・アームとマトリックスの乗算



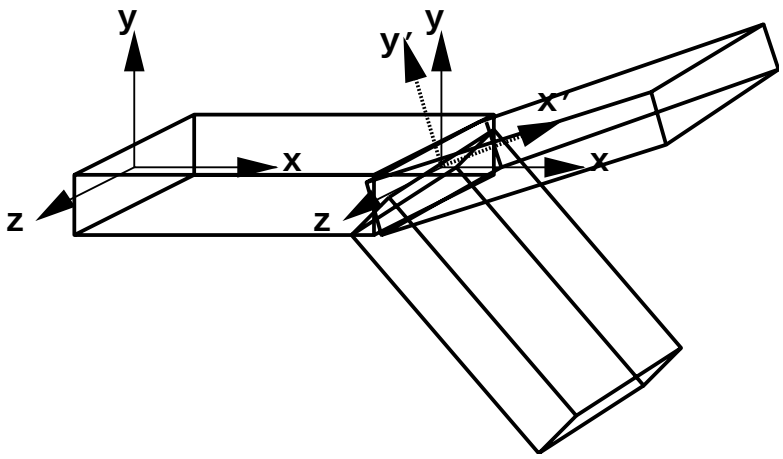
- モデリング変換を順に施してモデリング座標系を移動させていく
- モデリング変換の関数によって新たなモデリング変換のマトリックスがカレントのモデルビュー・マトリックスに掛け合わさっていく
- モデリング変換でモデリング座標系を移動させて描画すると上の図は簡単にプログラムできる
  1. 最初の腕を描く
  2. 肘の位置までモデリング座標系を移動させる (glTranslatef)
  3. Z 軸を回転の中心としてそのモデリング座標系を回転させる (glRotatef)
  4. 2 番目の腕を描く

多関節のアーム



- 1 番目と 2 番目の腕は前のアームと同じ
  1. 最初の腕を描く
  2. 肘の位置までモデリング座標系を移動させる
  3. Z 軸を回転の中心としてそのモデリング座標系を回転させる
  4. 2 番目の腕を描く
- 3 番目の腕はどう描くのか?
  5. 2 番目の肘の位置までモデリング座標系を移動する
  6. Z 軸を回転の中心としてそのモデリング座標系を回転させる
  7. 3 番目の腕を描く

一関節に複数の腕を持つアーム



- 2 番目の腕と 3 番目の腕が同じ肘で回転する
- 両方の腕は同じマトリックスに依存する
- glPushMatrix と glPopMatrix() で座標システムをセーブ/ロードする
- 2 番めの腕を描く前にマトリックスを保存
  1. 最初の腕を描く
  2. 肘の位置までモデリング座標系を移動させる
  3. カレント・マトリックスを保存する (glPushMatrix)
  4. Z 軸を回転の中心としてそのモデリング座標系を回転させる
  5. 2 番目の腕を描く
- 3 番目の腕を描く前にマトリックスを取り出す
  6. スタックから保存したマトリックスを取り出す (glPopMatrix)
  7. 1 番目の肘の位置で z 軸を中心にモデリング座標系を回転させる
  8. 3 番目の腕を描く

```
 1 /*
 2 ** modeling3.c - ロボットアーム
 3 */
 4
 5 #include <GL/glut.h>
 6 #include "laconia.h"
 7
 8 void
 9 initialize(void)
10 {
11   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
12
13   glMatrixMode( GL_MODELVIEW );
14   glLoadIdentity();
15   gluLookAt( 0.0, 50.0, 100.0,
16              0.0,  0.0,   0.0,
17              0.0,  1.0,   0.0 );
18 }
19
20 /* ロボットアームの描画 */
21 void
22 drawArm( float armAngle )
23 {
24   /* 1番目のアームの描画 */
25   lacArm( 20.0, 4.0, 8.0, 'w' );
26
27   /* モデリング座標系をヒジの位置に移動 */
28   glTranslatef( 20.0, 0.0, 0.0 );
29
30   /* 2番目のアームの回転、2番目のアームの描画 */
31   glRotatef( armAngle, 0.0, 0.0, 1.0 );
32   lacArm( 14.0, 4.0, 8.0, 'w' );
33 }
34
35 void
36 display(void)
37 {
38   /* モデリング変換のパラメータ */
39   static float offsetX = -51.0;
40   static float angle = -1.0;
41   static float armAngle = 9.0;
42
43   /* offsetXの計算 (-50.0~50.0) */
```

```
44   offsetX += 1.0;
45   if( offsetX>=51.0 ) offsetX -= 101.0;
46
47   /* angle の計算 (0.0~359.0) */
48   angle += 1.0;
49   if( angle>=360.0 ) angle -= 360.0;
50
51   /* armAngle の計算 (10.0~80.0) */
52   armAngle += 1.0;
53   if( armAngle>=81.0 ) armAngle -= 81.0;
54
55   glClear( GL_COLOR_BUFFER_BIT );
56
57   glPushMatrix();
58   glTranslatef( offsetX, 0.0, 0.0 );
59   glColor3f( 1.0f, 1.0f, 0.0f );
60   lacSphere( 10.0, 'w' );
61   glPopMatrix();
62
63   glPushMatrix();
64   glRotatef( angle, 0.0, 1.0, 0.0 );
65   glColor3f( 0.0f, 0.0f, 1.0f );
66   lacCone( 15.0, 15.0, 'w' );
67   glPopMatrix();
68
69   /* ロボットアームの描画 */
70   glPushMatrix();
71   glRotatef( -angle, 0.0, 1.0, 0.0 );
72   glColor3f( 1.0f, 1.0f, 1.0f );
73   drawArm( armAngle );
74   glPopMatrix();
75
76   glutSwapBuffers();
77 }
78
79 void
80 keyboard( unsigned char c, int x, int y )
81 {
82   if( c==27 ) exit( 0 );
83 }
84
85 void
86 reshape( int w, int h )
87 {
```

```
 88   glViewport( 0, 0, w, h );
 89
 90   glMatrixMode( GL_PROJECTION );
 91   glLoadIdentity();
 92   gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
 93   glMatrixMode( GL_MODELVIEW );
 94 }
 95
 96 void
 97 animate(void)
 98 {
 99   glutPostRedisplay();
100 }
101
102 int
103 main( int argc, char *argv[] )
104 {
105   glutInit( &argc, argv );
106
107   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE );
108   glutInitWindowPosition( 100, 100 );
109   glutInitWindowSize( 500, 400 );
110   glutCreateWindow( argv[0] );
111
112   glutDisplayFunc( display );
113   glutKeyboardFunc( keyboard );
114   glutReshapeFunc( reshape );
115   glutIdleFunc( animate );
116
117   initialize();
118   glutMainLoop();
119   return 0;
120 }
121
```

<演習>

1. forYou ディレクトリの modeling3 プログラムを実行して下さい。

```
% cd forYou
% make modeling3
% modeling3
```

2. animation プログラムに、modeling3 のロボットアームを組み込んで下さい。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

3. 2 番目のアームの先に 3 番目のアームを継ぎ足して下さい。
4. さらに 1 番目のアームの先に 4 番目のアームを継ぎ足して下さい。

この章の関数のまとめ

モデリング変換の指定

```
void glRotated(GLdouble angle, GLdouble x, GLdouble y, GLdouble z);
void glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z);
void glScaled(GLdouble x, GLdouble y, GLdouble z);
void glScalef(GLfloat x, GLfloat y, GLfloat z);
void glTranslated(GLdouble x, GLdouble y, GLdouble z);
void glTranslatef(GLfloat x, GLfloat y, GLfloat z);
```

変換マトリックスの操作

```
void glPushMatrix(void);
void glPopMatrix(void);

void glLoadMatrixf(GLfloat *m);
void glLoadMatrixd(GLdouble *m);
void glMultMatrixf(GLfloat *m);
void glMultMatrixd(GLdouble *m);
void glGetFloatv(GLenum pname,  GLfloat *m);
void glGetDoublev(GLenum pname,  GLdouble *m);
```

第5章　隠面消去

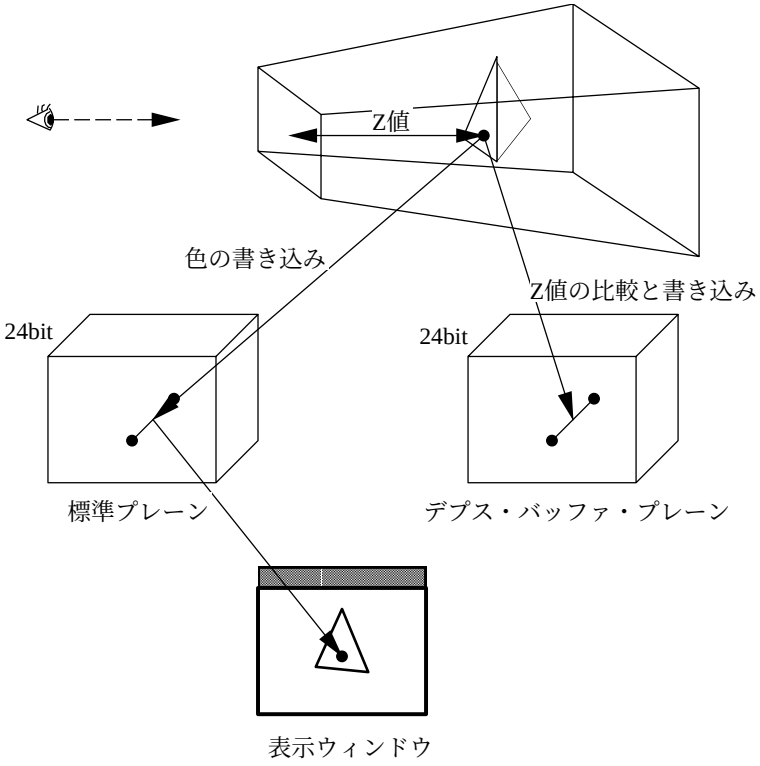
本章の目的

- 面を張った3次元物体を描画する
- ある面より後ろにあり、目（カメラ）から隠されている部分を表示しないようにプログラミングする
- デプス・バッファ・テストによる隠面消去を理解する
- バックフェーシング・ポリゴン・リムーバルによる隠面消去を理解する
- デプス・バッファ・テストとバックフェース・ポリゴン・リムーバルを組合わせて隠面消去の高速化を図る

5.1　デプス・バッファリング

デプス・バッファ・テストとは

- 表示される各点の輝度情報を格納する標準プレーンなどの他に、奥行き情報を格納するデプス・バッファ・プレーンを用意する
- 描画の前にデプス・バッファー・プレーンを一番遠い Z 値でクリアする
- 表示される各点に対して、奥行き z を計算して
  1. デプス・バッファ・プレーンの中にすでに格納されている z 値と比較する
  2. 新しい z 値の方が目に近い場合、その z 値をデプス・バッファ・プレーンの中に書き込む
     輝度情報 (新しい色) を標準プレーンなどに書き込む
  3. 新しい z 値の方が目から遠い場合、その新しい点は古い点の背後にあるので、その点は描かない
     デプス・バッファ・プレーンの内容は更新しない
- デプス・バッファは Z バッファとも呼ばれる



デプス・バッファ・サブルーチン(1)

glutInitDisplayMode (GLUT_SINGLE |GLUT_RGBA | GLUT_DEPTH);

隠面消去を使うときは必ず初期化時にGLUT_DEPTHを指定する
デプス・バッファ・テストはスクリーン上の各ピクセルについて深さ情報を格納するデプス・バッファが必要である
デプス・バッファ用のメモリの確保は各ウィンドウ・システムに依存する機能であるのでGLXやWGLなどのインターフェイス・ライブラリの役割である
ここではglutライブラリを用いる

glEnable(GL_DEPTH_TEST);

デプス・バッファ・テストによる隠面消去をオンにする

glDisable(GL_DEPTH_TEST);

デプス・バッファ・テストによる隠面消去をオフにする

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

標準プレーンとデプス・バッファ・プレーンを同時にクリアする

前章までは隠面消去の必要が無かったため標準プレーンのみをクリアしたが、3次元グラフィックスを扱うプログラムにおいてはデプス・バッファも同時にクリアする場合が多い

void glClearDepth(GLclampd depth);

glClearでデプス・バッファをクリアするときの値を設定する

通常はデフォルトの値(そのマシンで設定可能な最大値)のままでよいためこの関数を使うことは少ない

なお最大値が1.0、最小値が–1.0である

デプス・バッファ・サブルーチン (2)

void glDepthFunc(GLenum func);

デプス・バッファ・テストに用いる判定条件を指定する

GL_LEQUAL　新しい z 値が大きくなければ描かれる

GL_LESS　新しい z 値が小さければ描かれる

この判定条件がデフォルト

GL_GEQUAL　新しい z 値が小さくなければ描かれる

その他の GL_*　マニュアル・ページ参照

IRIS GL のデフォルトは LEQUAL で OpenGL と異なっているのに注意する
IRIS GL との互換性を維持したい場合にはこの関数を次のように使う

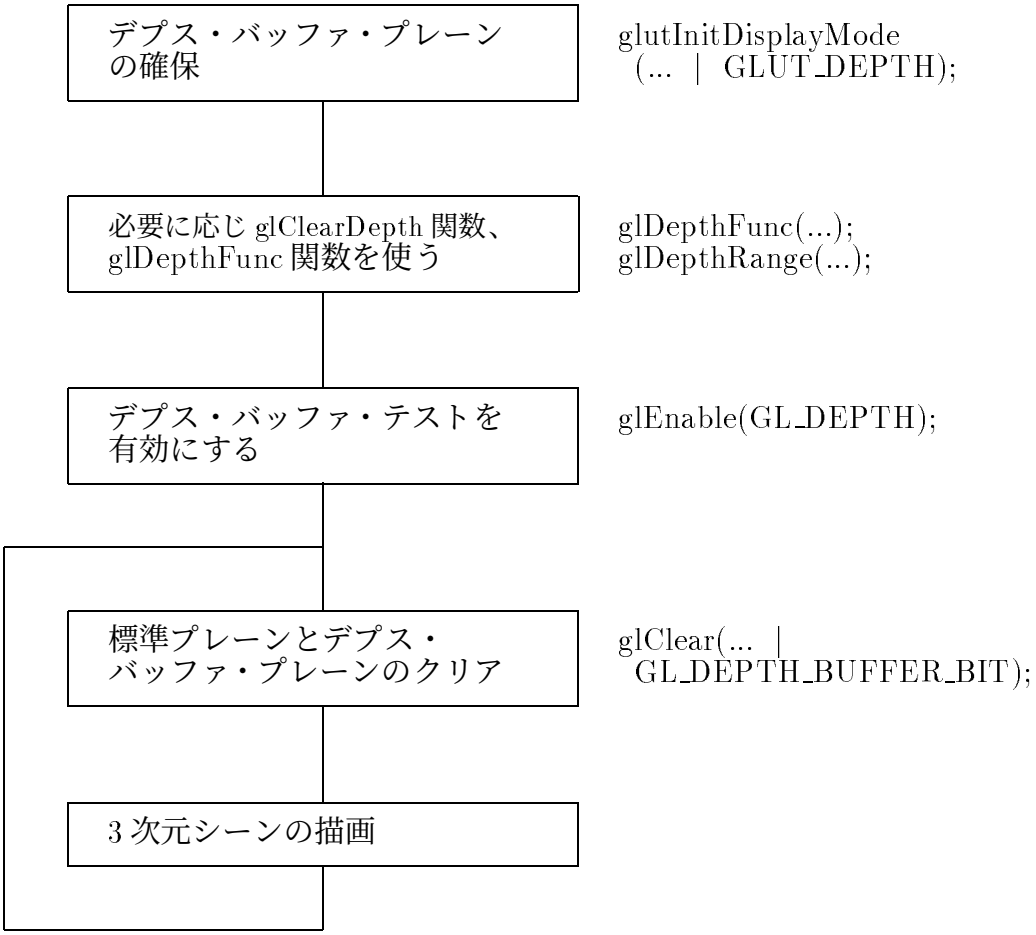
```
glDepthFunc(GL_LEQUAL);
```

void glDepthRange(GLclampd near, GLclampd far);

z 値の遠近のマッピングされる範囲を指定する

通常はデフォルトの [-1.0, 1.0] の範囲を使うので、この関数は使わなくてよい

デプス・バッファ・テストを使った プログラムの流れ



デプス・バッファリングの使い方

- C の場合

```
void
initWindow(void)
{
    ...
    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    ...
}

void
initOpenGL(void)
{
    ...
    /* glDepthFunc(GL_LEQUAL);  /* IRIS GL 互換にしたいとき */
    /* glDepthRange(-1.0, 1.0); /* 必要に応じて */
    /* glClearDepth(1.0);       /* 必要に応じて */
    glEnable(GL_DEPTH_TEST);
    ...
}

void
drawScene( ... )
{
    ...
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    /* シーンを描く */
    ...
}
```

隠面消去の加速化
– バックフェーシング・ポリゴンの利用 –

- デプス・バッファ・テストによる隠面消去にバックフェーシング・ポリゴン・リムーバルという手法を組み合わせるとさらに高速化できる
- この手法は物体の裏側に当たるポリゴン (back-facing polygon) を予め判定して描かないようにする
  - フロントフェーシング・ポリゴン
    頂点が投影面上（つまりディスプレイ上）で反時計回りに描かれているもの
  - バックフェーシング・ポリゴン
    頂点が投影面上（つまりディスプレイ上）で時計回りに描かれているもの
- バックフェーシング・ポリゴンによる方法は一つの閉凸多面体のみ完全な隠面消去ができる
  通常は単独で利用することはなくデプス・バッファ・テストと組み合わせる

- ビジュアル・シミュレーションの町並みのシーンのように、多数の閉凸多面体を組合わせたシーンを描画する場合、デプス・バッファ・テストと併用すると、ビルの背面を描画せずに済むため効果的に高速化できる

バックフェーシング・ポリゴン・リムーバルの使い方

- フロントフェーシング・ポリゴンがどちら向きかを決める

  glFrontFace(GLenum mode);

  mode は半時計回りのとき GL_CCW(デフォルト)、時計回りのとき GL_CW とする
- 描かない面を指定する

  glCullFace(GLenum mode);

  mode は GL_FRONT、GL_BACK(デフォルト)、GL_FRONT_AND_BACK
- バックフェーシング・ポリゴン・リムーバルのオン/オフ

  glEnable(GL_CULL_FACE);

  glDisable(GL_CULL_FACE);

```
 1 /*
 2 ** depth.c - デプスバッファリング
 3 */
 4
 5 #include <GL/glut.h>
 6 #include "laconia.h"
 7
 8 void
 9 initialize(void)
10 {
11   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
12
13   glMatrixMode( GL_MODELVIEW );
14   glLoadIdentity();
15   gluLookAt( 0.0, 50.0, 100.0,
16              0.0,  0.0,   0.0,
17              0.0,  1.0,   0.0 );
18
19   /* デプステストを有効にする */
20   glEnable( GL_DEPTH_TEST );
21
22   /* バックフェースポリゴンリムーバルを有効にする */
23   glEnable( GL_CULL_FACE );
24 }
25
26 void
27 drawArm( float armAngle )
28 {
29   lacArm( 20.0, 4.0, 8.0, 'w' );
30
31   glTranslatef( 20.0, 0.0, 0.0 );
32
33   glRotatef( armAngle, 0.0, 0.0, 1.0 );
34   lacArm( 14.0, 4.0, 8.0, 'w' );
35 }
36
37 void
38 display(void)
39 {
40   static float offsetX = -51.0;
41   static float angle = -1.0;
42   static float armAngle = 9.0;
43
```

```
44   offsetX += 1.0;
45   if( offsetX>=51.0 ) offsetX -= 101.0;
46   angle += 1.0;
47   if( angle>=360.0 ) angle -= 360.0;
48   armAngle += 1.0;
49   if( armAngle>=81.0 ) armAngle -= 81.0;
50
51   /* カラーバッファと共にデプスバッファもクリア */
52   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
53
54   glPushMatrix();
55   glTranslatef( offsetX, 0.0, 0.0 );
56   glColor3f( 1.0f, 1.0f, 0.0f );
57
58   /* 塗りつぶしモードで描画 */
59   lacSphere( 10.0, 's' );
60
61   glPopMatrix();
62
63   glPushMatrix();
64   glRotatef( angle, 0.0, 1.0, 0.0 );
65   glColor3f( 0.0f, 0.0f, 1.0f );
66
67   /* 塗りつぶしモードで描画 */
68   lacCone( 15.0, 15.0, 's' );
69
70   glPopMatrix();
71
72   glPushMatrix();
73   glRotatef( -angle, 0.0, 1.0, 0.0 );
74   glColor3f( 1.0f, 1.0f, 1.0f );
75   drawArm( armAngle );
76   glPopMatrix();
77
78   glutSwapBuffers();
79 }
80
81 void
82 keyboard( unsigned char c, int x, int y )
83 {
84   if( c==27 ) exit( 0 );
85 }
86
87 void
```

```
 88 reshape( int w, int h )
 89 {
 90   glViewport( 0, 0, w, h );
 91
 92   glMatrixMode( GL_PROJECTION );
 93   glLoadIdentity();
 94   gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
 95   glMatrixMode( GL_MODELVIEW );
 96 }
 97
 98 void
 99 animate(void)
100 {
101   glutPostRedisplay();
102 }
103
104 main( int argc, char *argv[] )
105 {
106   glutInit( &argc, argv );
107
108   /* デプスバッファの確保 */
109   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
110
111   glutInitWindowPosition( 100, 100 );
112   glutInitWindowSize( 500, 400 );
113   glutCreateWindow( argv[0] );
114
115   glutDisplayFunc( display );
116   glutKeyboardFunc( keyboard );
117   glutReshapeFunc( reshape );
118   glutIdleFunc( animate );
119
120   initialize();
121   glutMainLoop();
122 }
123
```

<演習>

1. forYou ディレクトリの depth プログラムを実行して下さい。

```
% cd forYou
% make depth
% depth
```

2. animation プログラムのワイヤーフレーム 3 次元モデルを、塗りつぶしモデルに変更して下さい。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

常に後から描画するものが上描きされることを確認して下さい。

3. animation プログラムにデプスバッファとバックフェーシングポリゴンリムーバルの処理を加え、適切な隠面消去が行なわれることを確認して下さい。

サブルーチンのまとめ

デプスバッファを使った隠面消去を行なう

```
void glFrontFace(GLenum mode);
void glEnable(GL_CULL_FACE);
void glDisable(GL_CULL_FACE);
```

デプスバッファによる隠面消去

```
void glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
void glEnable(GL_DEPTH_TEST);
void glDisable(GL_DEPTH_TEST);

void glDepthRange(GLclampd near, GLclampd far);
void glDepthFunc(GLenum func);
```

デプスバッファに関する情報の取得

```
void glGetIntegerv(GL_DEPTH_BITS, &param);
void glGetIntegerv(GL_DEPTH_CLEAR_VALUE, &param);
void glGetIntegerv(GL_DEPTH_FUNC, &param);
void glGetIntegerv(GL_DEPTH_RANGE, &param);
void glGetIntegerv(GL_DEPTH_TEST, &param);
```

デプスバッファをクリアする値を設定する

```
void glClearDepth(GLclampd depth);
```

デプスバッファのクリア

```
void glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

第6章　ディスプレイリスト

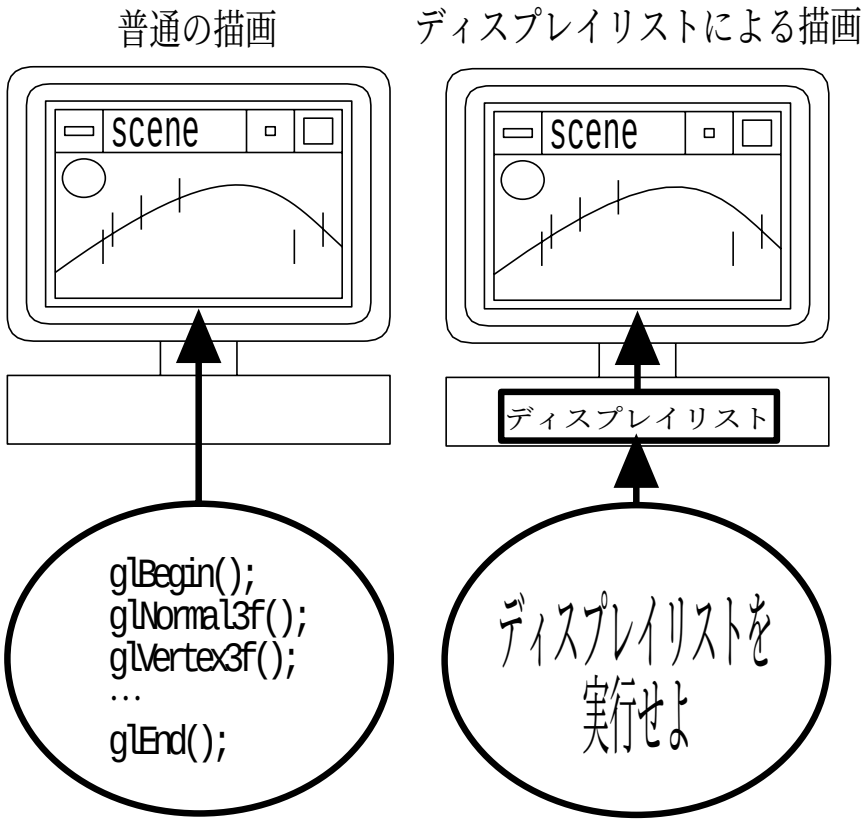
本章の目的

- ディスプレイリストの利点を理解する
- ディスプレイリストの使い方を習得する

6.1　ディスプレイリストの利点

ディスプレイリストとは

- グラフィックス命令のかたまり
- サーバーに蓄えられる
- 最初にディスプレイリスト作成時間がかかる
- 描画は高速



ディスプレイリストの特徴

- グラフィックス命令がサーバーに蓄えられるため、次のような場合に有利
  - グラフィックスが CPU より高速
  - ネットワークごしの描画
  - 複数のテクスチャーの切替え
  - 静的なモデルの描画
  - 階層構造をもったモデルの定義
- いくつかの OpenGL 命令はディスプレイリスト化できない

```
glIsList()
glGenLists()
glNewList()
glEndList()
glDeleteLists()
glFeedbackBuffer()
glSelectBuffer()
glRenderMode()
glReadPixels()
glPixelStore()
glFlush()
glFinish()
glIsEnabled()
glGet*()
```

6.2　ディスプレイリスト操作

ディスプレイリスト名

GLuint glGenLists(GLsizei count)

- 連続する空のディスプレイリスト名を生成する
  – 先頭リストのインデックス（名前）を返す
- count は生成するディスプレイリストの数
- ある名前がすでに使われているか調べることができる
  – GLboolean glIsList(GLuint list)

ディスプレイリストの作成

GLvoid glNewList(GLuint listIndex, GLenum mode)

GLvoid glEndList()

- listIndex はディスプレイリスト名
- mode は次のいずれか
  - GL_COMPILE
  - GL_COMPILE_AND_EXECUTE
- glNewList() と glEndList() の間の OpenGL ルーチン群が、ディスプレイリストとなる

```
glNewList( 1, GL_COMPILE );
  glColor3fv( colorVector );
  glTranslatef( 1.5, 0.0, 0.0 );
  glBegin( GL_TRIANGLES );
    glVertex2f( 0.0, 0.0 );
    glVertex2f( 1.0, 0.0 );
    glVertex2f( 0.0, 1.0 );
  glEnd();
glEndList();
```

- ディスプレイリスト化できないルーチンは、普通に実行される
  - ディスプレイリストに入らない
- ステート（状態）はディスプレイリスト前後で保存されない
  - 各種属性やマトリックスのプッシュ/ポップ操作はなされない

ディスプレイリストの実行

GLvoid glCallList(GLuint listIndex)

- １つのディスプレイリストを実行
- listIndex はディスプレイリスト名

複数のディスプレイリストの実行

GLvoid glCallLists(GLsizei count, GLenum type, const GLvoid *listIndices)

- １つ以上のディスプレイリストを実行
- count は実行するディスプレイリスト数
- listIndices はディスプレイリスト名の配列
- type は listIndices の型
  - GL_BYTE, GL_UNSIGNED_BYTE
  - GL_SHORT, GL_UNSIGNED_SHORT
  - GL_INT, GL_UNSIGNED_INT
  - GL_FLOAT
  - GL_2_BYTES, GL_3_BYTES, GL_4_BYTES
- 定義されていないディスプレイリストは無視される
- listIndices に対するオフセットを設定できる
  - GLvoid glListBase(GLuint base)
- 文字列などに多用される

ディスプレイリストの消去

GLvoid glDeleteLists(GLuint firstIndex, GLsizei count)

- １つ以上のディスプレイリストを消去
- firstIndex は消去するディスプレイリスト群の先頭のリスト名
- count は消去するディスプレイリスト数
- サーバーのメモリスペースを解放する
- ディスプレイリスト名が再利用できる
- 定義されていないディスプレイリスト名は無視される

ディスプレイリストの階層化

- ディスプレイリスト中に glCallList() をいれる

```
glNewList( 1, GL_COMPILE ); /* [1] を作成 */
  drawFin();
glEndList();

glNewList( 2, GL_COMPILE ); /* [2] を作成 */
  drawFish();
glEndList();

glNewList( 3, GL_COMPILE ); /* [3]（[1]、[2] を実行）を作成 */
  glCallList( 1 );
  glCallList( 2 );
glEndList();
   .
   .
   .
glCallList( 3 ); /* drawFin(); drawFish(); */
   .
   .
   .
```

- ディスプレイリストは編集できないが、以下のようにディスプレイリスト中のディスプレイリストを差し替えることができる

```
   .
   .
   .
glNewList( 1, GL_COMPILE ); /* [1] を再定義 */
  drawNewFin();
glEndList();
   .
   .
   .
glCallList( 3 ); /* drawNewFin(); drawFish(); */
   .
   .
   .
```

```
 1 /*
 2 ** displaylist.c - ディスプレイリスト
 3 */
 4
 5 #include <GL/glut.h>
 6 #include <math.h>
 7 #include "laconia.h"
 8 #include "oglportable.h"
 9
10 /* ディスプレイリスト用の静的変数を確保 */
11 static GLuint index;
12
13 /* 山を描画するディスプレイリストの生成 */
14 void
15 createMountainList( float size )
16 {
17   int i, j, k;
18   double angle;
19   float radius, x, y, z;
20
21   /* ディスプレイリスト名の生成 */
22   index = glGenLists( 1 );
23
24   /* ディスプレイリスト作成開始 */
25   glNewList( index, GL_COMPILE_AND_EXECUTE );
26
27   for( i=0; i<6; i++ )
28   {
29     glBegin(GL_QUAD_STRIP);
30     for( j=0; j<37; j++ )
31     {
32       angle = 2.0*M_PI*(double)j/36.0;
33       for( k=0; k<2; k++ )
34       {
35         radius = size*(float)(5+k-i)/6.0;
36
37         x = radius*(float)cos(angle);
38         y = (1.0+(float)cos(M_PI*(double)(radius/size)))*size/2.0;
39         z = -radius*(float)sin(angle);
40         glVertex3f(x, y, z);
41       }
42     }
43     glEnd();
```

```
44   }
45
46   /* ディスプレイリスト作成終了 */
47   glEndList();
48
49   return;
50 }
51
52 void
53 initialize(void)
54 {
55   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
56
57   glMatrixMode( GL_MODELVIEW );
58   glLoadIdentity();
59   gluLookAt( 0.0, 50.0, 100.0,
60              0.0,  0.0,   0.0,
61              0.0,  1.0,   0.0 );
62
63   glEnable( GL_DEPTH_TEST );
64   glEnable( GL_CULL_FACE );
65
66   /* ディスプレイリストの生成 */
67   createMountainList( 20.0f );
68 }
69
70 void
71 display(void)
72 {
73   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
74
75   glPushMatrix();
76   glColor3f( 1.0f, 1.0f, 0.0f );
77
78   /* ディスプレイリストの描画 */
79   glCallList( index );
80
81   glPopMatrix();
82
83   glutSwapBuffers();
84 }
85
86 void
```

```
 87 keyboard( unsigned char c, int x, int y )
 88 {
 89   if( c==27 ) exit( 0 );
 90 }
 91
 92 void
 93 reshape( int w, int h )
 94 {
 95   glViewport( 0, 0, w, h );
 96
 97   glMatrixMode( GL_PROJECTION );
 98   glLoadIdentity();
 99   gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
100   glMatrixMode( GL_MODELVIEW );
101 }
102
103 void
104 animate(void)
105 {
106   glutPostRedisplay();
107 }
108
109 int
110 main( int argc, char *argv[] )
111 {
112   glutInit( &argc, argv );
113
114   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
115
116   glutInitWindowPosition( 100, 100 );
117   glutInitWindowSize( 500, 400 );
118   glutCreateWindow( argv[0] );
119
120   glutDisplayFunc( display );
121   glutKeyboardFunc( keyboard );
122   glutReshapeFunc( reshape );
123   glutIdleFunc( animate );
124
125   initialize();
126   glutMainLoop();
127   return 0;
128 }
129
```

<演習>

1. forYou ディレクトリの displaylist プログラムを実行して下さい。

```
% cd forYou
% make displaylist
% displaylist
```

2. animation プログラムのモデルのいずれかを、ディスプレイリスト化して下さい。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

3. animation プログラムに、ディスプレイリストを使ったモデルを追加して下さい。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

サブルーチンのまとめ

ディスプレイリスト名の生成

```
GLuint glGenLists( GLsizei count )
```

ディスプレイリストの作成

```
GLvoid glNewList( GLuint listIndex, GLenum mode )
GLvoid glEndList()
```

ディスプレイリストの実行

```
GLvoid glCallList( GLuint listIndex )
GLvoid glCallLists( GLsizei count, GLenum type, const GLvoid *listIndices )
```

ディスプレイリストの削除

```
GLvoid glDeleteLists( GLuint firstIndex, GLsizei count )
```

第7章　ライティング基礎

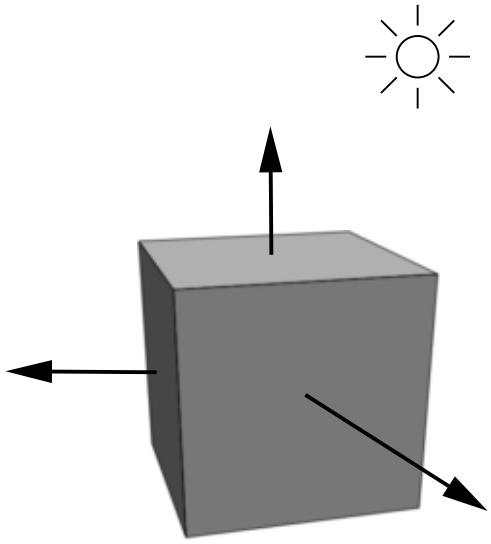
本章の目的

- ライティングの概要を把握する
- OpenGL のデフォルト設定を使ったライティングを行なう
- 法線ベクトルの作成と設定を習得する
- 基本的なライティングを行なう

7.1　ライティング概要

ライティングとは

- ライティングとは、モデルの色を OpenGL の光源処理機能で自動生成し、モデルに光をあてた効果を実現すること
- 点や線、ポリゴンの色は自動生成される
  – glColor*() は不要
- 色の自動生成のため、次の 3 つが必要
  – 光源処理機能をオンにする
  – 1 つ以上の光源をオンにする
  – 法線を指定しながらモデルを描画
- OpenGL は光源処理を行なう際、次の 3 種類のライティングプロパティを参照する
  – マテリアル（質感）プロパティ
  – ライト（光源）プロパティ
  – ライトモデル（環境）プロパティ
- 本章では 3 種類のライティングプロパティについて、OpenGL のデフォルト設定を使う



ライティングのオン/オフ

GLvoid glEnable(GLenum option)

GLvoid glDisable(GLenum option)

- option は GL_LIGHTING
- ライティングプロパティを用い、頂点や面の色を自動生成する

光源のオン/オフ

GLvoid glEnable(GLenum option)

GLvoid glDisable(GLenum option)

- option は GL_LIGHTn
  - n は 0〜（最大の光源数–1）
- 最大の光源数を調べておく
  - glGetIntegerv(GL_MAX_LIGHTS, &numLights)
- デフォルトの光源は、–Z 方向への平行光

7.2 法線ベクトル

法線とは

- 法線とは、面に垂直なベクトルのこと
- モデル頂点の色を自動生成するために、頂点ごとの情報を与える
- glColor*() で色指定する代わりに、法線を与える

法線の指定

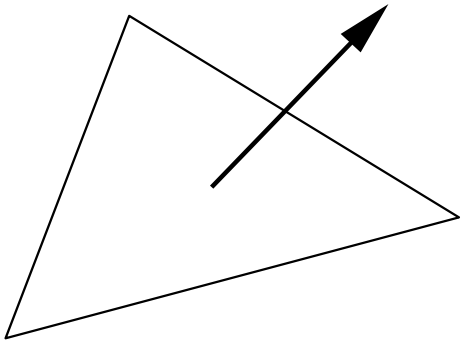
GLvoid glNormal3f(GLfloat nx, GLfloat ny, GLfloat nz)

- カレント法線を（nx,ny,nz）にセットする
  – 続く glVertex*() で指定される頂点は、この法線を使う
- 法線は正規化（長さが 1）されていなければならない
- 法線は、面の表裏も決定する
- 引数指定にスカラー型とベクター型がある

スカラー型	ベクター型
glNormal3b(nx, ny, nz)	glNormal3bv(*v)
glNormal3s(nx, ny, nz)	glNormal3sv(*v)
glNormal3i(nx, ny, nz)	glNormal3iv(*v)
glNormal3f(nx, ny, nz)	glNormal3fv(*v)
glNormal3d(nx, ny, nz)	glNormal3dv(*v)

面の法線

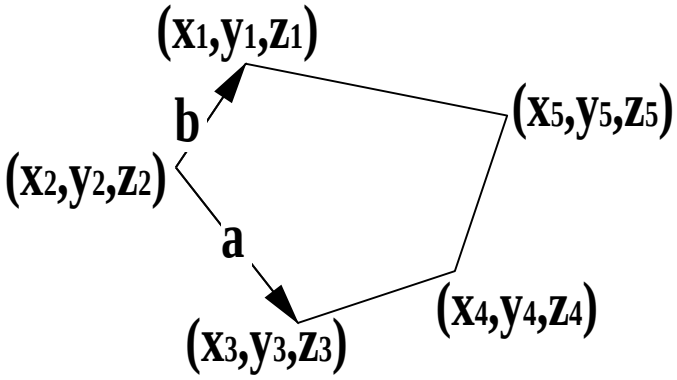
- 全ての頂点が同じ法線をもつとき、それを面の法線と呼ぶ



- ポリゴンの塗りつぶしはフラットシェーディング
- ポリゴンの頂点座標値から、面の法線が外積で求まる
  – 全ての頂点が同一平面上にあると仮定

面の法線の計算

1. 隣合う３つの頂点を選択



例えば (x_1, y_1, z_1)、(x_2, y_2, z_2)、(x_3, y_3, z_3) を選ぶとする

2. a、b の２ベクトルを計算

$$(a_x, a_y, a_z) = (x_3 - x_2,\ y_3 - y_2,\ z_3 - z_2)$$

$$(b_x, b_y, b_z) = (x_1 - x_2,\ y_1 - y_2,\ z_1 - z_2)$$

3. a、b ベクトルの外積を計算

$$n_x = a_y b_z - a_z b_y$$
$$n_y = a_z b_x - a_x b_z$$
$$n_z = a_x b_y - a_y b_x$$

4. 法線を正規化

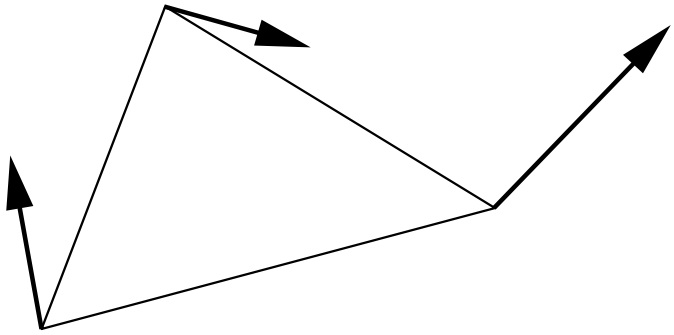
$$length = \sqrt{n_x^2 + n_y^2 + n_z^2}$$
$$N_x = n_x \,/\, length$$
$$N_y = n_y \,/\, length$$
$$N_z = n_z \,/\, length$$

5. ポリゴンに対する法線を指定

```
glNormal3f( Nx, Ny, Nz );
glBegin( GL_POLYGON );
  glVertex3fv( v1 );
  glVertex3fv( v2 );
  glVertex3fv( v3 );
  glVertex3fv( v4 );
  glVertex3fv( v5 );
glEnd();
```

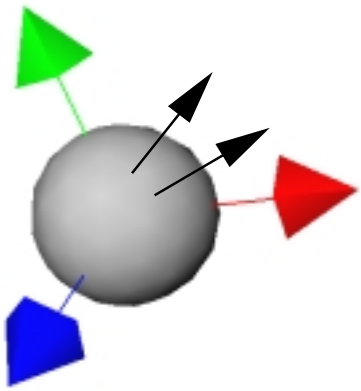
頂点の法線

- 頂点が各々法線をもつとき、それを頂点の法線と呼ぶ



- ポリゴンの塗りつぶしはグーローシェーディング
  – 曲面（球など）に使う
- 頂点の法線を求めるには、次の 2 つの手法がある
  – 数学的手法
  – 汎用的手法

頂点法線の計算（数学的手法）

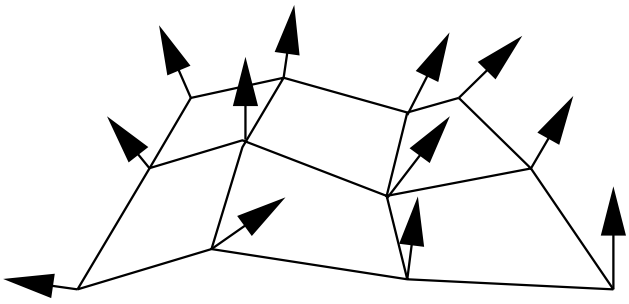


頂点の法線が数学的に計算できる場合は…

1. 各頂点の法線を計算
2. 法線を正規化
3. 法線を使う

例：座標原点を中心とする球では、頂点座標値をそのまま法線ベクトルとして使える

頂点法線の計算（汎用的手法）



頂点の法線が数学的に計算できない場合は…

1. 全ポリゴンの法線を計算
2. 各頂点について…
   (a) その頂点を使っている全てのポリゴンを探す（m 個あるとする）
   (b) m 個のポリゴンの法線の和を求める

$$n_x = nx_1 + nx_2 + nx_3 + \ ... \ + nx_m$$

$$n_y = ny_1 + ny_2 + ny_3 + \ ... \ + ny_m$$

$$n_z = nz_1 + nz_2 + nz_3 + \ ... \ + nz_m$$

   (c) 法線の和を正規化
   (d) 法線を使う

法線自動正規化のオン/オフ

GLvoid glEnable(GLenum option)

GLvoid glDisable(GLenum option)

- option は GL_NORMALIZE
- 必要となるのは…
  - モデルの拡大縮小を行なうとき
  - 正規化していない法線を使うとき
- 性能が劣化する

```
 1 /*
 2 ** light.c - 基本的なライティング
 3 */
 4
 5 #include <GL/glut.h>
 6 #include <math.h>
 7 #include "laconia.h"
 8 #include "oglportable.h"
 9
10 static GLuint index;
11
12 void
13 createMountainList( float size )
14 {
15   int i, j, k;
16   double angle;
17   float radius, iSlant;
18   float x, y, z;
19   float nx, ny, nz;
20
21   index = glGenLists( 1 );
22
23   glNewList( index, GL_COMPILE_AND_EXECUTE );
24
25   for( i=0; i<6; i++ )
26   {
27     glBegin(GL_QUAD_STRIP);
28     for( j=0; j<37; j++ )
29     {
30       angle = 2.0*M_PI*(double)j/36.0;
31       for( k=0; k<2; k++ )
32       {
33         radius = size*(float)(5+k-i)/6.0;
34
35         iSlant = (float)sin(M_PI*(double)(radius/size));
36         nx = (iSlant==0.0)? 0.0: (float)cos( angle );
37         ny = (iSlant==0.0)? 1.0: 1.0/iSlant;
38         nz = (iSlant==0.0)? 0.0: (float)-sin( angle );
39
40         /* 法線ベクトルの設定 */
41         glNormal3f( nx, ny, nz );
42
43         x = radius*(float)cos(angle);
```

```
44          y = (1.0+(float)cos(M_PI*(double)(radius/size)))*size/2.0;
45          z = -radius*(float)sin(angle);
46          glVertex3f(x, y, z);
47        }
48      }
49      glEnd();
50    }
51
52    glEndList();
53
54    return;
55  }
56
57  void
58  initialize(void)
59  {
60    glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
61
62    glMatrixMode( GL_MODELVIEW );
63    glLoadIdentity();
64    gluLookAt( 0.0, 50.0, 100.0,
65               0.0,  0.0,   0.0,
66               0.0,  1.0,   0.0 );
67
68    glEnable( GL_DEPTH_TEST );
69    glEnable( GL_CULL_FACE );
70
71    createMountainList( 20.0f );
72
73    /* ライティングを有効にする */
74    glEnable( GL_LIGHTING );
75
76    /* 光源を1つ有効にする */
77    glEnable( GL_LIGHT0 );
78
79    /* 法線ベクトルの自動正規化を有効にする */
80    glEnable( GL_NORMALIZE );
81  }
82
83  void
84  display(void)
85  {
86    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
```

```
 87
 88   /* 色の指定は無効なため削除 */
 89 /*
 90   glColor3f( 1.0f, 1.0f, 0.0f );
 91 */
 92   glCallList( index );
 93
 94   glutSwapBuffers();
 95 }
 96
 97 void
 98 keyboard( unsigned char c, int x, int y )
 99 {
100   if( c==27 ) exit( 0 );
101 }
102
103 void
104 reshape( int w, int h )
105 {
106   glViewport( 0, 0, w, h );
107
108   glMatrixMode( GL_PROJECTION );
109   glLoadIdentity();
110   gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
111   glMatrixMode( GL_MODELVIEW );
112 }
113
114 void
115 animate(void)
116 {
117   glutPostRedisplay();
118 }
119
120 int
121 main( int argc, char *argv[] )
122 {
123   glutInit( &argc, argv );
124
125   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
126
127   glutInitWindowPosition( 100, 100 );
128   glutInitWindowSize( 500, 400 );
129   glutCreateWindow( argv[0] );
130
```

```
131   glutDisplayFunc( display );
132   glutKeyboardFunc( keyboard );
133   glutReshapeFunc( reshape );
134   glutIdleFunc( animate );
135
136   initialize();
137   glutMainLoop();
138   return 0;
139 }
140
```

<演習>

1. forYou ディレクトリの light プログラムを実行して下さい。

```
% cd forYou
% make light
% light
```

2. animation プログラムを、光源処理を使ったプログラムに変更して下さい。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

サブルーチンのまとめ

ライティングのオン/オフ

```
GLvoid glEnable( GL_LIGHTING )
GLvoid glDisable( GL_LIGHTING )
```

光源のオン/オフ

```
GLvoid glEnable( GL_LIGHTn )
GLvoid glDisable( GL_LIGHTn )
```

法線の指定

```
GLvoid glNormal3f( GLfloat nx, GLfloat ny, GLfloat nz )
```

第8章　ライティングプロパティ

本章の目的

- マテリアル（質感）プロパティを理解する
- 基本的なライト（光源）プロパティを理解する
- ライトモデル（環境）プロパティを理解する
- 3 種のプロパティの設定方法を習得する

8.1　ライティングプロパティ概要

ライティングプロパティの種類

- ライティングは、次の情報からモデルの頂点色を自動生成する
  – マテリアルプロパティ - モデル表面の質感属性
  – ライトプロパティ - 光源の属性
  – ライトモデルプロパティ - 光源とモデルをとりまく環境の属性

ライティングプロパティのセーブ/リストア

GLvoid glPushAttrib(GLbitfield mask)

GLvoid glPopAttrib(GLvoid)

- mask は GL_LIGHTING_BIT
- glPushAttrib() はカレントのライティングプロパティをセーブ
- glPopAttrib() はセーブされたプロパティをリストア

8.2　マテリアルプロパティ

マテリアルプロパティの設定

GLvoid glMaterialfv(GLenum face, GLenum pname, const GLfloat *params)

- face は次のいずれかで、どの面にプロパティを設定するか指定する
  - GL_FRONT
  - GL_BACK
  - GL_FRONT_AND_BACK
- pname はマテリアルプロパティ名
- params はプロパティ配列
- マテリアルプロパティに次のものがある

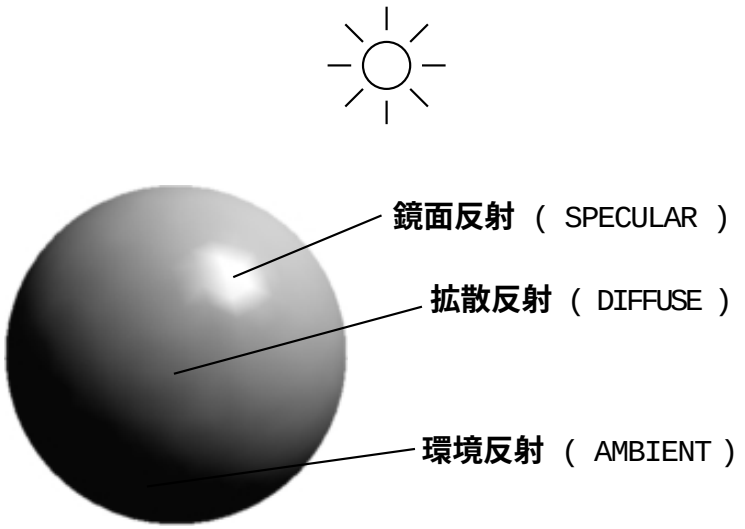
	pname	デフォルト
反射色	GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)
	GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)
	GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)
	GL_AMBIENT_AND_DIFFUSE	-
反射強度	GL_SHININESS	0.0
発色	GL_EMISSION	(0.0, 0.0, 0.0, 1.0)

- 引数指定にスカラー型とベクター型がある

スカラー型	ベクター型
glMaterialf(face, pname, val)	glMaterialfv(face, pname, *v)
glMateriali(face, pname, val)	glMaterialiv(face, pname, *v)

各種マテリアルプロパティ

- 鏡面反射
  - pname は GL_SPECULAR
  - params は 0.0〜1.0 で RGBA の輝度
  - デフォルトは（0.0, 0.0, 0.0, 1.0）
- 拡散反射
  - pname は GL_DIFFUSE
  - params は 0.0〜1.0 で RGBA の輝度
  - デフォルトは（0.8, 0.8, 0.8, 1.0）
- 環境反射
  - pname は GL_AMBIENT
  - params は 0.0 〜1.0 で RGBA の輝度
  - デフォルトは（0.2, 0.2, 0.2, 1.0）
- 拡散反射と環境反射プロパティを同時にセットできる
  - pname は GL_AMBIENT_AND_DIFFUSE



- 反射強度
  - 鏡面反射の強度
  - pname は GL_SHININESS
  - params は 0.0〜128.0
    0.0 - 鏡面反射無効
    1.0〜128.0 - 大きな値で鏡面反射部分が小さく、強くなる
  - デフォルトは 0.0
- 発色
  - pname は GL_EMISSION
  - params は 0.0〜1.0 で RGBA の輝度
  - デフォルトは（0.0, 0.0, 0.0, 1.0）

```
 1 /*
 2 ** material.c - マテリアルプロパティの設定
 3 */
 4
 5 #include <GL/glut.h>
 6 #include "laconia.h"
 7
 8 void
 9 initialize(void)
10 {
11   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
12
13   glMatrixMode( GL_MODELVIEW );
14   glLoadIdentity();
15   gluLookAt( 0.0, 50.0, 100.0,
16              0.0,  0.0,   0.0,
17              0.0,  1.0,   0.0 );
18
19   glEnable( GL_DEPTH_TEST );
20   glEnable( GL_CULL_FACE );
21
22   glEnable( GL_LIGHTING );
23   glEnable( GL_LIGHT0 );
24   glEnable( GL_NORMALIZE );
25 }
26
27 void
28 display(void)
29 {
30   static GLfloat ad[] = { 0.0f, 1.0f, 0.0f, 1.0f };
31   static GLfloat shininess[] = { 50.0f };
32
33   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
34
35   /* 色指定の代わりにマテリアルプロパティを設定する */
36   glMaterialfv( GL_FRONT, GL_AMBIENT_AND_DIFFUSE, ad );
37   glMaterialfv( GL_FRONT, GL_SHININESS, shininess );
38
39   lacMountain( 20.0 );
40
41   glutSwapBuffers();
42 }
43
```

```
44 void
45 keyboard( unsigned char c, int x, int y )
46 {
47   if( c==27 ) exit( 0 );
48 }
49
50 void
51 reshape( int w, int h )
52 {
53   glViewport( 0, 0, w, h );
54
55   glMatrixMode( GL_PROJECTION );
56   glLoadIdentity();
57   gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
58   glMatrixMode( GL_MODELVIEW );
59 }
60
61 void
62 animate(void)
63 {
64   glutPostRedisplay();
65 }
66
67 int
68 main( int argc, char *argv[] )
69 {
70   glutInit( &argc, argv );
71
72   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
73
74   glutInitWindowPosition( 100, 100 );
75   glutInitWindowSize( 500, 400 );
76   glutCreateWindow( argv[0] );
77
78   glutDisplayFunc( display );
79   glutKeyboardFunc( keyboard );
80   glutReshapeFunc( reshape );
81   glutIdleFunc( animate );
82
83   initialize();
84   glutMainLoop();
85   return 0;
86 }
87
```

<演習>

1. forYou ディレクトリの material プログラムを実行して下さい。

```
% cd forYou
% make material
% material
```

2. animation プログラムのいくつかのモデルに、マテリアルプロパティを与えて下さい。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

8.3　基本ライトプロパティ

ライトプロパティの設定

GLvoid glLightfv(GLenum light, GLenum pname, const GLfloat *params)

- light は GL_LIGHTn
  – n は 0〜（最大の光源数–1）
- pname はライトプロパティ名
- params はプロパティ配列
- 基本ライトプロパティに次のものがある

	pname	デフォルト
光源色	GL_SPECULAR	(1.0, 1.0, 1.0, 1.0) - GL_LIGHT0 (0.0, 0.0, 0.0, 1.0) - GL_LIGHT1〜
	GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0) - GL_LIGHT0 (0.0, 0.0, 0.0, 1.0) - GL_LIGHT1〜
	GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)
光源位置	GL_POSITION	(0.0, 0.0, 1.0, 0.0)

- 引数指定にスカラー型とベクター型がある

スカラー型	ベクター型
glLightf(light, pname, val)	glLightfv(light, pname, *v)
glLighti(light, pname, val)	glLightiv(face, pname, *v)

各種基本ライトプロパティ

- 鏡面光
  - モデルの鏡面反射色
  - pname は GL_SPECULAR
  - params は 0.0〜1.0 で RGBA の輝度
  - デフォルトは、GL_LIGHT0 に対して（1.0, 1.0, 1.0, 1.0）、それ以外は（0.0, 0.0, 0.0, 1.0）
- 拡散光
  - モデルの拡散反射色
  - pname は GL_DIFFUSE
  - params は 0.0〜1.0 で RGBA の輝度
  - デフォルトは、GL_LIGHT0 に対して（1.0, 1.0, 1.0, 1.0）、それ以外は（0.0, 0.0, 0.0, 1.0）
- 環境光
  - 光源が与えるシーンへの環境光
  - pname は GL_AMBIENT
  - params は 0.0〜1.0 で RGBA の輝度
  - デフォルトは（0.0, 0.0, 0.0, 1.0）
- 光源位置
  - pname は GL_POSITION
  - params は xyzw の座標値
  - w=0.0 は無限遠光源（x、y、z から原点への平行光）
  - w=1.0 はローカル光源（x、y、z は光源位置）
  - デフォルトは（0.0, 0.0, 1.0, 0.0）
  - 無限遠光源は、ローカル光源より高速

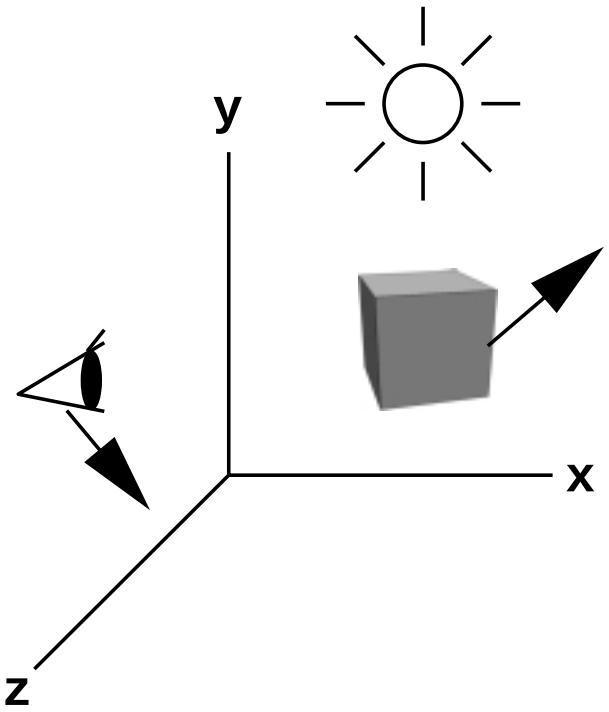
光源位置とモデルビューマトリックス

- 光源位置は glLightfv(...,GL_POSITION,...) 時に、モデルビューマトリックスで移動する
- 光源の移動は、シーン中のモデルと同様に考えることができる
  – ただし光源は実際のモデルより先に指定
- 光源位置の制御法は、いくつかある
  – シーン中で移動しない光源
  – モデルのように移動する光源
  – 視点に追従する光源

静的な光源

- 光源が、シーン中で移動しない
- glLightfv() をビューイング変換直後に行なう

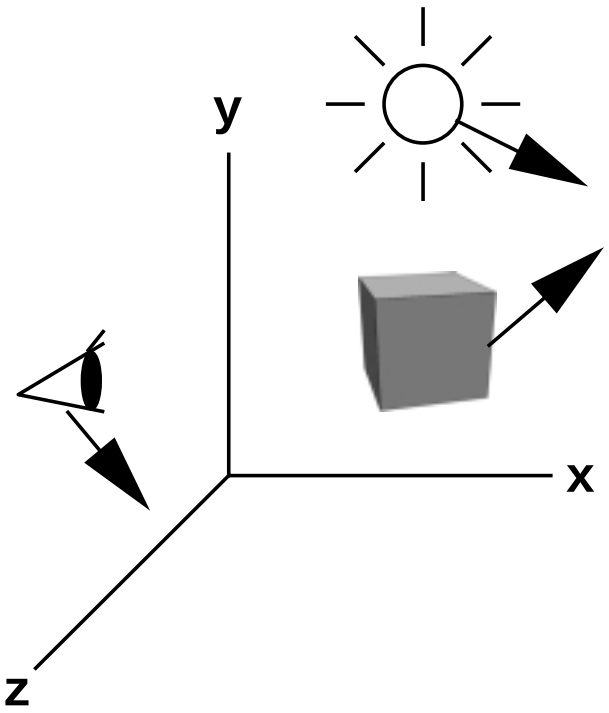
```
glPushMatrix();
  gluLookAt(...);
  glLightfv( GL_LIGHT0, GL_POSITION, lPos );
  drawScene();
glPopMatrix();
```



モデルのように移動する光源

- モデリング変換による光源移動
- 光源用のモデリング変換を行なう

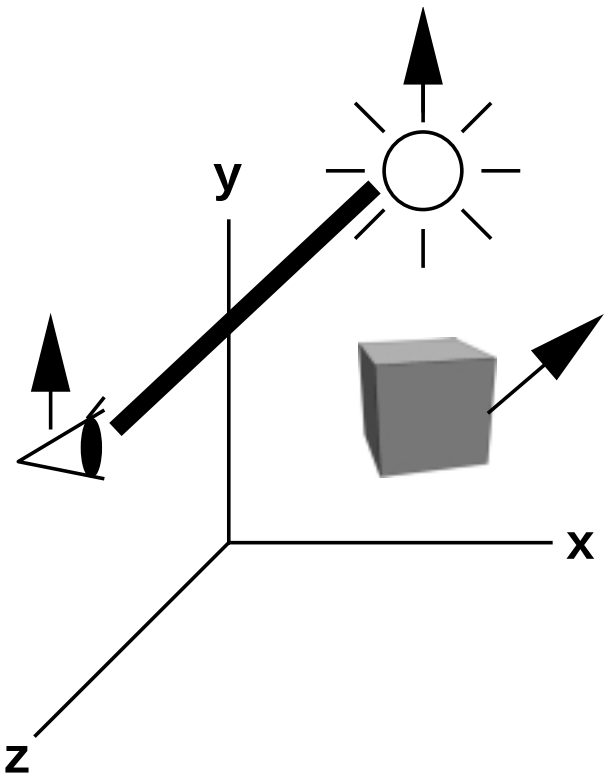
```
glPushMatrix();
  gluLookAt(...);
  glPushMatrix();
    glTranslatef(...);
    glRotatef(...);
    glLightfv( GL_LIGHT0, GL_POSITION, lPos );
  glPopMatrix();
  drawScene();
glPopMatrix();
```



視点に追従する光源

- glLightfv() をビューイング変換直前に行なう

```
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
gluPerspective(...);
glMatrixMode( GL_MODELVIEW );
glLightfv( GL_LIGHT0, GL_POSITION, lPos );
gluLookAt(...);
```



```
 1 /*
 2 ** movelight.c - 移動光源
 3 */
 4
 5 #include <GL/glut.h>
 6 #include "laconia.h"
 7
 8 void
 9 initialize(void)
10 {
11   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
12
13   glMatrixMode( GL_MODELVIEW );
14   glLoadIdentity();
15   gluLookAt( 0.0, 50.0, 100.0,
16              0.0,  0.0,   0.0,
17              0.0,  1.0,   0.0 );
18
19   glEnable( GL_DEPTH_TEST );
20   glEnable( GL_CULL_FACE );
21
22   glEnable( GL_LIGHTING );
23   glEnable( GL_LIGHT0 );
24   glEnable( GL_NORMALIZE );
25 }
26
27 void
28 display(void)
29 {
30   static GLfloat ad[] = { 0.0f, 1.0f, 0.0f, 1.0f };
31   static GLfloat shininess[] = { 50.0f };
32   static GLfloat lightPos[] = { 0.0f, 0.0f, 0.0f, 1.0f };
33
34   /* 光源位置パラメーター */
35   static float lightX = -51.0f;
36
37   /* 光源位置の計算 */
38   lightX += 1.0f;
39   if( lightX>=51.0f ) lightX -= 101.0f;
40
41   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
42
43   glPushMatrix();
```

```
44
45   /* 光源移動用モデリング変換 */
46   glTranslatef( lightX, 30.0f, 0.0f );
47
48   /* 光源のジオメトリ描画と位置設定 */
49   glDisable( GL_LIGHTING );
50   lacSphere( 1.0, 's' );
51   glEnable( GL_LIGHTING );
52   glLightfv( GL_LIGHT0, GL_POSITION, lightPos );
53
54   glPopMatrix();
55
56   glMaterialfv( GL_FRONT, GL_AMBIENT_AND_DIFFUSE, ad );
57   glMaterialfv( GL_FRONT, GL_SHININESS, shininess );
58   lacMountain( 20.0 );
59
60   glutSwapBuffers();
61 }
62
63 void
64 keyboard( unsigned char c, int x, int y )
65 {
66   if( c==27 ) exit( 0 );
67 }
68
69 void
70 reshape( int w, int h )
71 {
72   glViewport( 0, 0, w, h );
73
74   glMatrixMode( GL_PROJECTION );
75   glLoadIdentity();
76   gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
77   glMatrixMode( GL_MODELVIEW );
78 }
79
80 void
81 animate(void)
82 {
83   glutPostRedisplay();
84 }
85
86 int
87 main( int argc, char *argv[] )
```

```
 88 {
 89   glutInit( &argc, argv );
 90
 91   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
 92
 93   glutInitWindowPosition( 100, 100 );
 94   glutInitWindowSize( 500, 400 );
 95   glutCreateWindow( argv[0] );
 96
 97   glutDisplayFunc( display );
 98   glutKeyboardFunc( keyboard );
 99   glutReshapeFunc( reshape );
100   glutIdleFunc( animate );
101
102   initialize();
103   glutMainLoop();
104   return 0;
105 }
106
```

<演習>

1. forYou ディレクトリの movelight プログラムを実行して下さい。

```
% cd forYou
% make movelight
% movelight
```

2. animation プログラムの光源を、動くようにして下さい。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

8.4　ライトモデルプロパティ

ライトモデルプロパティの設定

GLvoid glLightModelfv(GLenum pname, const GLfloat *params)

- ライトモデルは、マテリアルやライトで設定できない環境属性を決める
- pname はライトモデルプロパティ名
- params はプロパティ配列
- ライトモデルプロパティに次のものがある

	pname	デフォルト
環境光	GL_LIGHT_MODEL_AMBIENT	(0.2, 0.2, 0.2, 1.0)
視点位置	GL_LIGHT_MODEL_LOCAL_VIEWER	0.0
両面フラグ	GL_LIGHT_MODEL_TWO_SIDE	0.0

- 引数指定にスカラー型とベクター型がある

スカラー型	ベクター型
glLightModelf(pname, val)	glLightModelfv(pname, *v)
glLightModeli(pname, val)	glLightModeliv(pname, *v)

各種ライトモデルプロパティ

- 環境光
  - シーン全体に与える環境色
  - pname は GL_LIGHT_MODEL_AMBIENT
  - params は 0.0〜1.0 で RGBA の輝度
  - デフォルトは（0.2, 0.2, 0.2, 1.0）
- 視点位置フラグ
  - 視点位置を無限遠、ローカルどちらと仮定して計算するか指定する
  - 鏡面反射の効果に影響する
  - pname は GL_LIGHT_MODEL_LOCAL_VIEWER
  - デフォルトは 0.0（無限遠視点）
- 両面ライティングフラグ
  - バックフェースポリゴンの法線の向きが、自動的に逆になる
  - 線や点には無効
  - 表と裏に異なるマテリアルプロパティをセットできる
  - pname は GL_LIGHT_MODEL_TWO_SIDE
  - デフォルトは 0.0（表だけのライティング）

```
 1 /*
 2 ** twoside.c - 両面ライティング
 3 */
 4
 5 #include <GL/glut.h>
 6 #include "laconia.h"
 7
 8 void
 9 initialize(void)
10 {
11   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
12
13   glMatrixMode( GL_MODELVIEW );
14   glLoadIdentity();
15   gluLookAt( 0.0, 50.0, 100.0,
16              0.0,  0.0,   0.0,
17              0.0,  1.0,   0.0 );
18
19   glEnable( GL_DEPTH_TEST );
20   /* バックフェースポリゴンリムーバルを削除 */
21 /*
22   glEnable( GL_CULL_FACE );
23 */
24
25   glEnable( GL_LIGHTING );
26   glEnable( GL_LIGHT0 );
27   glEnable( GL_NORMALIZE );
28
29   /* 両面ライティングを有効にする */
30   glLightModelf( GL_LIGHT_MODEL_TWO_SIDE, 1.0 );
31 }
32
33 void
34 display(void)
35 {
36   static GLfloat front[] = { 0.0, 1.0, 0.0, 1.0 };
37   static GLfloat back[] = { 0.0, 0.0, 1.0, 1.0 };
38   static GLfloat lightPos[] = { 0.0, 0.0, 0.0, 1.0 };
39   static float lightX = -51.0;
40   static float angle = -1.0;
41
42   lightX += 1.0;
43   if( lightX>=51.0 ) lightX -= 101.0;
```

```
44   angle += 1.0;
45   if( angle>=360.0 ) angle -= 360.0;
46
47   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
48
49   glPushMatrix();
50   glTranslatef( lightX, 30.0, 0.0 );
51   glDisable( GL_LIGHTING );
52   lacSphere( 1.0, 's' );
53   glEnable( GL_LIGHTING );
54   glLightfv( GL_LIGHT0, GL_POSITION, lightPos );
55   glPopMatrix();
56
57   glPushMatrix();
58   glRotatef( -angle, 1.0, 0.0, 0.0 );
59
60   /* ポリゴンの表裏に別々の色を設定 */
61   glMaterialfv( GL_FRONT, GL_AMBIENT_AND_DIFFUSE, front );
62   glMaterialfv( GL_BACK, GL_AMBIENT_AND_DIFFUSE, back );
63
64   lacMountain( 20.0 );
65   glPopMatrix();
66
67   glutSwapBuffers();
68 }
69
70 void
71 keyboard( unsigned char c, int x, int y )
72 {
73   if( c==27 ) exit( 0 );
74 }
75
76 void
77 reshape( int w, int h )
78 {
79   glViewport( 0, 0, w, h );
80
81   glMatrixMode( GL_PROJECTION );
82   glLoadIdentity();
83   gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
84   glMatrixMode( GL_MODELVIEW );
85 }
86
87 void
```

```
 88 animate(void)
 89 {
 90   glutPostRedisplay();
 91 }
 92
 93 int
 94 main( int argc, char *argv[] )
 95 {
 96   glutInit( &argc, argv );
 97
 98   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
 99
100   glutInitWindowPosition( 100, 100 );
101   glutInitWindowSize( 500, 400 );
102   glutCreateWindow( argv[0] );
103
104   glutDisplayFunc( display );
105   glutKeyboardFunc( keyboard );
106   glutReshapeFunc( reshape );
107   glutIdleFunc( animate );
108
109   initialize();
110   glutMainLoop();
111   return 0;
112 }
113
```

<演習>

1. forYou ディレクトリの twoside プログラムを実行して下さい。

```
% cd forYou
% make twoside
% twoside
```

2. animation プログラム中の 1 枚ポリゴンの裏表に、別々の色を与えて下さい。
glDisable(GL_CULL_FACE)、glEnable(GL_CULL_FACE) を、両面ライティングを施すモデルの描画前後で指定して下さい。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

サブルーチンのまとめ

ライティングプロパティのセーブ/リストア

```
GLvoid glPushAttrib( GLbitfield GL_LIGHTING_BIT )
GLvoid glPopAttrib( void )
```

マテリアルプロパティの設定

```
GLvoid glMaterialfv( GLenum face, GLenum pname, const GLfloat *params )
```

ライトプロパティの設定

```
GLvoid glLightfv( GLenum light, GLenum pname, const GLfloat *params )
```

ライトモデルプロパティの設定

```
GLvoid glLightModelfv( GLenum pname, const GLfloat *params )
```

第9章　ライティング上級

- マテリアルプロパティの高速な変更方法を習得する
- 光の減衰率を設定する
- スポットライティングを使う

9.1　高速なマテリアル変更

glColor*() によるマテリアルの変更

GLvoid glColorMaterial(GLenum face, GLenum property)

- 同一のマテリアルプロパティを変更する場合、glMaterial*() よりも高速
- face は次のいずれか
  - GL_FRONT
  - GL_BACK
  - GL_FRONT_AND_BACK（デフォルト）
- property は次のいずれか
  - GL_DIFFUSE
  - GL_AMBIENT
  - GL_SPECULAR
  - GL_EMISSION
  - GL_AMBIENT_AND_DIFFUSE（デフォルト）

  のいずれか
- glColor*() は、property で指定されたマテリアルプロパティを変更する

高速なマテリアル変更のオン/オフ

GLvoid glEnable(GLenum option)

GLvoid glDisable(GLenum option)

- option は GL_COLOR_MATERIAL
- 高速なマテリアル変更を用いた例

```
GLfloat red[]  = { 1.0, 0.0, 0.0 };
GLfloat blue[] = { 0.0, 1.0, 0.0 };

glColorMaterial( GL_FRONT, GL_AMBIENT_AND_DIFFUSE );
glEnable( GL_COLOR_MATERIAL );

glColor3fv( red );
drawRedModel();
glColor3fv( blue );
drawBlueModel();

glDisable( GL_COLOR_MATERIAL );
```

- glColor*() によるマテリアルの変更が終了したら、GL_COLOR_MATERIAL を無効にする

```
 1 /*
 2 ** colormaterial.c - 高速なマテリアルの切替え
 3 */
 4
 5 #include <GL/glut.h>
 6 #include "laconia.h"
 7
 8 void
 9 initialize(void)
10 {
11   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
12
13   glMatrixMode( GL_MODELVIEW );
14   glLoadIdentity();
15   gluLookAt( 0.0, 50.0, 100.0,
16              0.0,  0.0,   0.0,
17              0.0,  1.0,   0.0 );
18
19   glEnable( GL_DEPTH_TEST );
20   glEnable( GL_CULL_FACE );
21
22   glEnable( GL_LIGHTING );
23   glEnable( GL_LIGHT0 );
24   glEnable( GL_NORMALIZE );
25
26   /* glColor*() で切り替えられるパラメータを指定する */
27   glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
28
29   /* glColor*() によるマテリアルの切替えを有効にする */
30   glEnable( GL_COLOR_MATERIAL );
31 }
32
33 void
34 display(void)
35 {
36   static GLfloat lightPos[] = { 0.0f, 0.0f, 0.0f, 1.0f };
37   static float lightX = -51.0f;
38
39   lightX += 1.0f;
40   if( lightX>=51.0f ) lightX -= 101.0f;
41
42   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
43
```

```
44   glPushMatrix();
45   glTranslatef( lightX, 30.0f, 0.0f );
46   glDisable( GL_LIGHTING );
47   glColor3f( 1.0f, 1.0f, 1.0f );
48   lacSphere( 1.0, 's' );
49   glEnable( GL_LIGHTING );
50   glLightfv( GL_LIGHT0, GL_POSITION, lightPos );
51   glPopMatrix();
52
53   glPushMatrix();
54   glTranslatef( 20.0f, 0.0f, 0.0f );
55
56   /* マテリアルプロパティの設定 */
57   glColor3f( 1.0f, 0.0f, 0.0f );
58
59   lacSphere( 15.0, 's' );
60   glPopMatrix();
61
62   glPushMatrix();
63   glTranslatef( -20.0f, 0.0f, 0.0f );
64
65   /* マテリアルプロパティの設定 */
66   glColor3f( 0.0f, 1.0f, 0.0f );
67
68   lacMountain( 20.0 );
69   glPopMatrix();
70
71   glutSwapBuffers();
72 }
73
74 void
75 keyboard( unsigned char c, int x, int y )
76 {
77   if( c==27 ) exit( 0 );
78 }
79
80 void
81 reshape( int w, int h )
82 {
83   glViewport( 0, 0, w, h );
84
85   glMatrixMode( GL_PROJECTION );
86   glLoadIdentity();
```

```
 87   gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
 88   glMatrixMode( GL_MODELVIEW );
 89 }
 90
 91 void
 92 animate(void)
 93 {
 94   glutPostRedisplay();
 95 }
 96
 97 int
 98 main( int argc, char *argv[] )
 99 {
100   glutInit( &argc, argv );
101
102   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
103
104   glutInitWindowPosition( 100, 100 );
105   glutInitWindowSize( 500, 400 );
106   glutCreateWindow( argv[0] );
107
108   glutDisplayFunc( display );
109   glutKeyboardFunc( keyboard );
110   glutReshapeFunc( reshape );
111   glutIdleFunc( animate );
112
113   initialize();
114   glutMainLoop();
115   return 0;
116 }
117
```

<演習>

1. forYou ディレクトリの colormaterial プログラムを実行して下さい。

```
% cd forYou
% make colormaterial
% colormaterial
```

2. animation プログラムの各モデルの色を、glColor*() で設定するようにして下さい。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

9.2　上級ライトプロパティ

上級ライトプロパティの設定

GLvoid glLightfv(GLenum light, GLenum pname, const GLfloat *params)

- light は GL_LIGHTn
- pname はライトプロパティ名
- params はプロパティ配列
- 上級ライトプロパティに次のものがある

	pname	デフォルト
減衰率	GL_CONSTANT_ATTENUATION	k0 = 1.0
	GL_LINEAR_ATTENUATION	k1 = 0.0
	GL_QUADRATIC_ATTENUATION	k2 = 0.0
スポットライテイング	GL_SPOT_DIRECTION	(0.0, 0.0, −1.0)
	GL_SPOT_CUTOFF	180.0
	GL_SPOT_EXPONENT	0.0

減衰率とは

- 光源からの距離による明るさの減衰率
- 実世界では…

$$減衰率 = 1/(k2 \cdot distance^2)$$

- OpenGL では…

$$減衰率 = 1/(k0 + k1 \cdot distance + k2 \cdot distance^2)$$

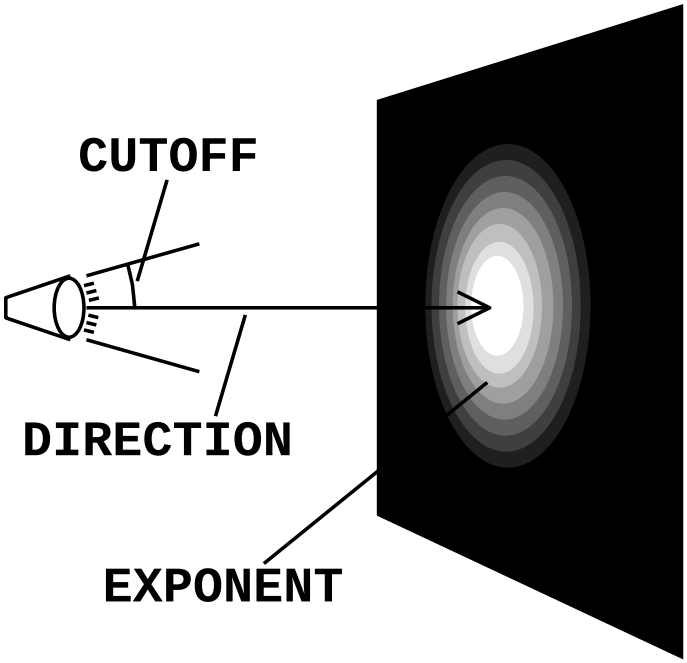
  - k0 は GL_CONSTANT_ATTENUATION
  - k1 は GL_LINEAR_ATTENUATION
  - k2 は GL_QUADRATIC_ATTENUATION
- ローカル光源の、環境光、拡散光、鏡面光プロパティに対して有効

減衰率の設定

- デフォルトは
  - k0 = 1.0（GL_CONSTANT_ATTENUATION）
  - k1 = 0.0（GL_LINEAR_ATTENUATION）
  - k2 = 0.0（GL_QUADRATIC_ATTENUATION）
- 距離が 10 のところで 50% の減衰率を実現するには…
  - k1 を利用： k0 = 1.0、k1 = 0.1、k2 = 0.0
  - k2 を利用： k0 = 1.0、k1 = 0.0、k2 = 0.01
- k0 は、光源の場所（距離 0）の減衰率を決定する
  - k0 に 0.0 を設定してはいけない
- 減衰率の設定により、描画速度が落ちる

スポットライトの作成

- glLight*() でスポットライトを作る
- GL_POSITION でローカル光源を指定
  - w = 1.0
  - モデルビューマトリックスの影響（位置）を受ける
- GL_SPOT_DIRECTION で光線の向きを指定
  - 光軸の向き（ベクトル）を、x、y、z で指定
  - 自動的に正規化される
  - モデルビューマトリックスの影響（向き）を受ける
  - デフォルトは（0.0, 0.0, –1.0）
- GL_SPOT_CUTOFF で照らす範囲を指定
  - 0.0〜90.0 で、光軸からの角度を指定
  - 180.0 は、点光源でスポットライトの効果がオフとなる
  - デフォルトは 180.0
- GL_SPOT_EXPONENT で、光軸からの角度による光の減衰率を指定
  - 0.0 で、範囲内一定の明るさ
  - 1.0〜128.0 で、減衰率指定
  - 1.0 では、光軸との角度の cos で減衰
  - 128.0 で最も鋭く減衰
  - デフォルトは 0.0


CUTOFF
DIRECTION
EXPONENT

```
 1 /*
 2 ** spotlight.c - スポットライト
 3 */
 4
 5 #include <GL/glut.h>
 6 #include "laconia.h"
 7
 8 void
 9 initialize(void)
10 {
11   static GLfloat spotDirection[] = { 0.0f, -1.0f, 0.0f };
12   static GLfloat spotCutoff[] = { 40.0f };
13
14   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
15
16   glMatrixMode( GL_MODELVIEW );
17   glLoadIdentity();
18   gluLookAt( 0.0, 50.0, 100.0,
19              0.0,  0.0,   0.0,
20              0.0,  1.0,   0.0 );
21
22   glEnable( GL_DEPTH_TEST );
23   glEnable( GL_CULL_FACE );
24
25   glEnable( GL_LIGHTING );
26   glEnable( GL_LIGHT0 );
27   glEnable( GL_NORMALIZE );
28
29   glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
30   glEnable( GL_COLOR_MATERIAL );
31
32   /* LIGHT_0をスポットライトにする */
33   glLightfv( GL_LIGHT0, GL_SPOT_DIRECTION, spotDirection );
34   glLightfv( GL_LIGHT0, GL_SPOT_CUTOFF, spotCutoff );
35 }
36
37 void
38 display(void)
39 {
40   static GLfloat lightPos[] = { 0.0f, 0.0f, 0.0f, 1.0f };
41   static float lightX = -51.0f;
42
43   lightX += 1.0f;
```

```
44   if( lightX>=51.0f ) lightX -= 101.0f;
45
46   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
47
48   glPushMatrix();
49   glTranslatef( lightX, 30.0f, 0.0f );
50   glDisable( GL_LIGHTING );
51   glColor3f( 1.0f, 1.0f, 1.0f );
52   lacSphere( 1.0, 's' );
53   glEnable( GL_LIGHTING );
54   glLightfv( GL_LIGHT0, GL_POSITION, lightPos );
55   glPopMatrix();
56
57   glPushMatrix();
58   glTranslatef( 20.0f, 0.0f, 0.0f );
59   glColor3f( 1.0f, 0.0f, 0.0f );
60   lacSphere( 15.0, 's' );
61   glPopMatrix();
62
63   glPushMatrix();
64   glTranslatef( -20.0f, 0.0f, 0.0f );
65   glColor3f( 0.0f, 1.0f, 0.0f );
66   lacMountain( 20.0 );
67   glPopMatrix();
68
69   glutSwapBuffers();
70 }
71
72 void
73 keyboard( unsigned char c, int x, int y )
74 {
75   if( c==27 ) exit( 0 );
76 }
77
78 void
79 reshape( int w, int h )
80 {
81   glViewport( 0, 0, w, h );
82
83   glMatrixMode( GL_PROJECTION );
84   glLoadIdentity();
85   gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
86   glMatrixMode( GL_MODELVIEW );
```

```
 87 }
 88
 89 void
 90 animate(void)
 91 {
 92   glutPostRedisplay();
 93 }
 94
 95 int
 96 main( int argc, char *argv[] )
 97 {
 98   glutInit( &argc, argv );
 99
100   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
101
102   glutInitWindowPosition( 100, 100 );
103   glutInitWindowSize( 500, 400 );
104   glutCreateWindow( argv[0] );
105
106   glutDisplayFunc( display );
107   glutKeyboardFunc( keyboard );
108   glutReshapeFunc( reshape );
109   glutIdleFunc( animate );
110
111   initialize();
112   glutMainLoop();
113   return 0;
114 }
115
```

<演習>

1. forYou ディレクトリの spotlight プログラムを実行して下さい。

```
% cd forYou
% make spotlight
% spotlight
```

2. animation プログラムの光源をスポットライトにして下さい。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

サブルーチンのまとめ

高速なマテリアル変更の設定

```
GLvoid glColorMaterial( GLenum face, GLenum property )
```

高速なマテリアル変更のオン/オフ

```
GLvoid glEnable( GL_COLOR_MATERIAL )
GLvoid glDisable( GL_COLOR_MATERIAL )
```

ライトプロパティの設定

```
GLvoid glLightfv( GLenum light, GLenum pname, const GLfloat *params )
```

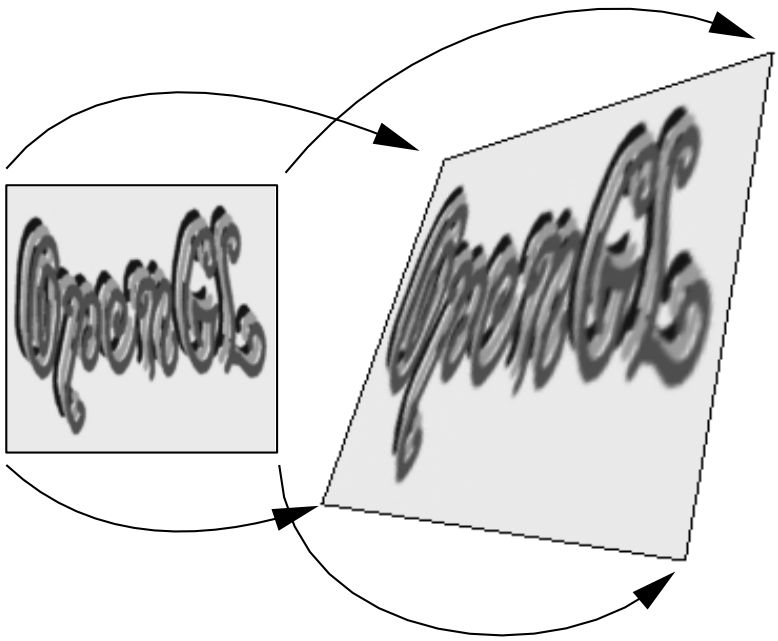
第10章　テクスチャーマッピング基礎

本章の目的

- テクスチャー座標の概念を把握する
- テクスチャーコンポーネントの概念を把握する
- テクスチャーのセットアップを習得する
- 基本的なテクスチャーマッピングを行なう

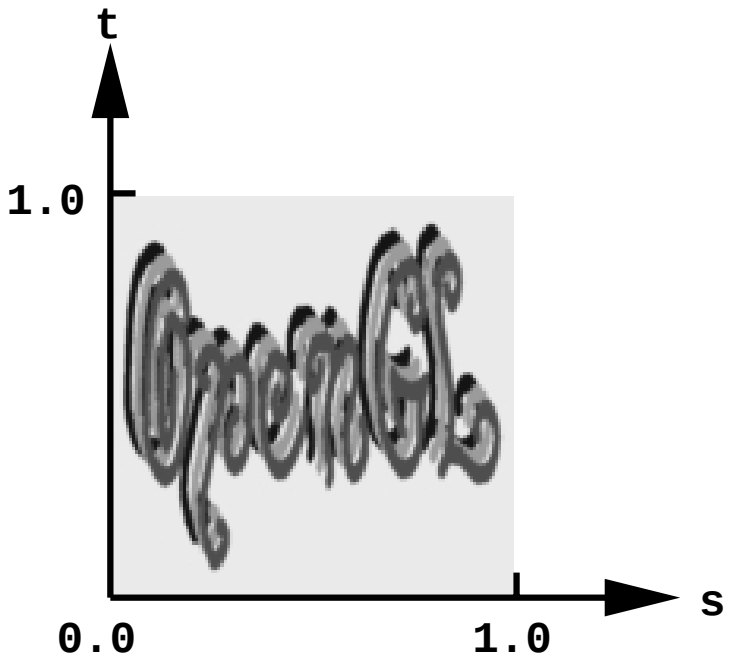
10.1　テクスチャーマッピングとは

- ラスターイメージをポリゴンにはり付けること



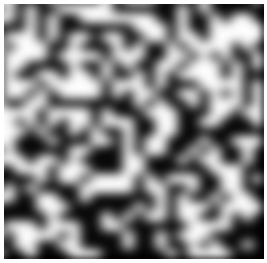
テクスチャーとは

- テクスチャーは、ピクセル情報の配列
  - 通常２次元
  - １次元の場合もある
- 通常ラスターイメージが使われる
- テクスチャーピクセルを「テクセル」と呼ぶ
- テクスチャーを定義する座標空間は、２次元の s-t 空間
- s、t が 0.0〜1.0 の範囲でテクスチャーを定義する
- ポリゴンの各頂点はテクスチャー座標値をもつ
- ポリゴンの塗りつぶしは、テクスチャーを参照しながら行なわれる

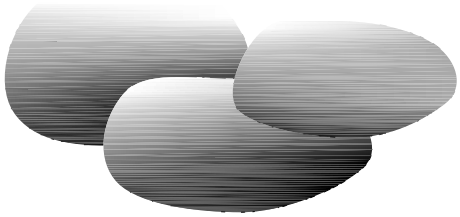


テクスチャーのコンポーネント

- １テクセルあたり１～４つのコンポーネント
- １コンポーネントテクスチャー
  – １コンポーネントで輝度を表す
  – 木材、芝生、砂地など



- ２コンポーネントテクスチャー
  – ２コンポーネントで輝度と透明度を表す
  – 樹木、雲など



- ３コンポーネントテクスチャー
  – ３コンポーネントで赤緑青を表す

- ４コンポーネントテクスチャー
  – ４コンポーネントで赤緑青、透明度を表す

- コンポーネント数が小さいほどテクスチャー転送速度は高速

テクスチャーマッピングの実現

- テクスチャーマッピングは、次のステップで実現する
  1. テクスチャーのセットアップ
  2. テクスチャー環境のセットアップ
     （モデルオリジナル色とテクスチャー色のブレンド法、など）
  3. 各頂点にテクスチャー座標値を与えながらモデルを描画
- ステップ２はデフォルトを使える
- ステップ３のテクスチャー座標値は、自動生成できる

10.2 テクスチャーのセットアップ

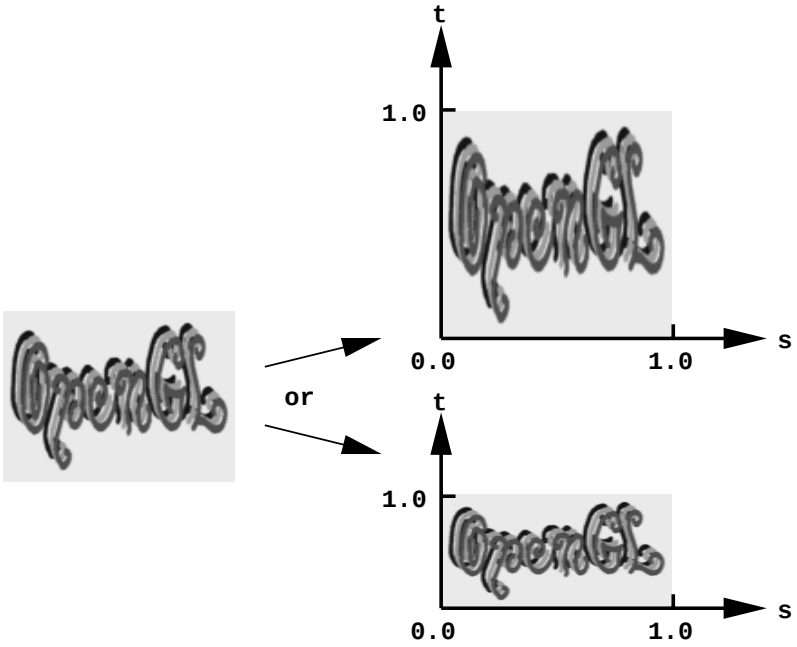
テクスチャーの生成

GLvoid glTexImage2D(GLenum target, GLint level, GLint components, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *image)

- target はターゲットテクスチャー
  - 常に GL_TEXTURE_2D
- level はミップマップレベル
  - 0 はベースイメージレベル
  - i は i 番目のミップマップイメージ
- components はカラーコンポーネント数
  - 1〜4 のいずれか
- width はテクスチャーの幅
  - $2^m + 2 \times border$（m は自然数）
- height はテクスチャーの高さ
  - $2^n + 2 \times border$（n は自然数）
- border は枠の幅
  - 0、1 のいずれか
- format はテクセルフォーマット
  - GL_COLOR_INDEX、GL_RED、GL_GREEN、GL_BLUE、GL_ALPHA、GL_RGB、GL_RGBA、GL_LUMINANCE、GL_LUMINANCE_ALPHA のいずれか
- type はテクセルタイプ
  - GL_UNSIGNED_BYTE、GL_BYTE、GL_BITMAP、GL_UNSIGNED_SHORT、GL_SHORT、GL_UNSIGNED_INT、GL_INT、GL_FLOAT のいずれか
- image はメモリ上のイメージ配列
  - イメージは常に $2^m \times 2^n$
  - 大きなイメージは転送時間がかかる

不適切なサイズのイメージ

- glTexImage2D() に使うイメージは $2^m \times 2^n$ でなければいけない
- OpenGL ユーティリティルーチンでイメージの拡大縮小ができる
  – gluScaleImage()
- glTexImage2D() には適切なサイズに拡大縮小したものを使う



テクスチャーの拡大縮小

GLint gluScaleImage(GLenum format, GLsizei widthin, GLint heightin, GLenum typein, const void *datain, GLint widthout, GLint heightout, GLenum typeout, void *dataout)

- format はピクセルフォーマット
  - GL_COLOR_INDEX、GL_STENCIL_INDEX、GL_DEPTH_COMPONENT、GL_RED、GL_GREEN、GL_BLUE、GL_ALPHA、GL_RGB、GL_RGBA、GL_LUMINANCE、GL_LUMINANCE_ALPHA のいずれか
- widthin、heightin は元のイメージサイズ
- typein は datain のピクセルタイプ
  - GL_UNSIGNED_BYTE、GL_BYTE、GL_BITMAP、GL_UNSIGNED_SHORT、GL_SHORT、GL_UNSIGNED_INT、GL_INT、GL_FLOAT のいずれか
- datain は元のイメージ配列
- widthout、heightout は拡大縮小後のイメージサイズ
- typeout は dataout のピクセルタイプ
  - GL_UNSIGNED_BYTE、GL_BYTE、GL_BITMAP、GL_UNSIGNED_SHORT、GL_SHORT、GL_UNSIGNED_INT、GL_INT、GL_FLOAT のいずれか
- dataout は拡大縮小後のイメージ配列

テクスチャーのオン/オフ

GLvoid glEnable(GLenum option)

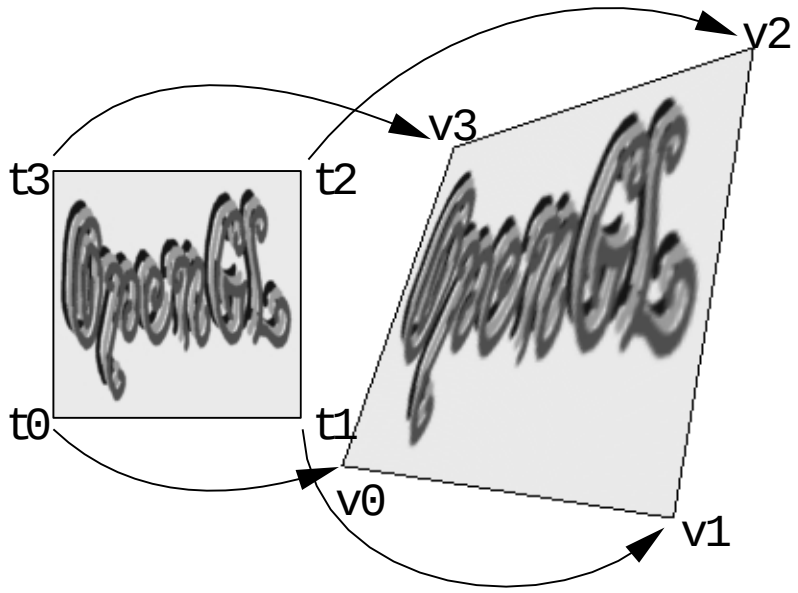
GLvoid glDisable(GLenum option)

- option は GL_TEXTURE_2D
- カレントのテクスチャー、テクスチャー座標値を使ってポリゴンを描画
- テクスチャーのないポリゴンを描画するときにはオフにする

テクスチャーポリゴンの描画

GLvoid glTexCoord2fv(const GLfloat *v)

- カレントのテクスチャー座標値を設定する
- v はテクスチャー座標値




```
glBegin( GL_QUADS );
  glTexCoord2fv( t0 );
  glVertex3fv( v0 );
  glTexCoord2fv( t1 );
  glVertex3fv( v1 );
  glTexCoord2fv( t2 );
  glVertex3fv( v2 );
  glTexCoord2fv( t3 );
  glVertex3fv( v3 );
glEnd();
```

```
 1 /*
 2 ** texture.c - 基本的なテクスチャーマッピング
 3 */
 4
 5 #include <GL/glut.h>
 6 #include "laconia.h"
 7 #include "sgiimage.h"
 8
 9 void
10 initialize(void)
11 {
12   unsigned int* image;
13   GLsizei       width, height;
14
15   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
16
17   glMatrixMode( GL_MODELVIEW );
18   glLoadIdentity();
19   gluLookAt( 0.0, 50.0, 100.0,
20              0.0,  0.0,   0.0,
21              0.0,  1.0,   0.0 );
22
23   glEnable( GL_DEPTH_TEST );
24   glEnable( GL_CULL_FACE );
25
26   /* テクスチャーのセットアップ */
27   image = rgbReadImageFile( "../images/ocean.rgb",
28                             &width, &height );
29   glTexImage2D( GL_TEXTURE_2D, 0, 3, width, height,
30                 0, GL_RGBA, GL_UNSIGNED_BYTE, image );
31
32   /* 縮小フィルターの設定（次章で詳述）*/
33   glTexParameterf( GL_TEXTURE_2D,
34                    GL_TEXTURE_MIN_FILTER, GL_LINEAR );
35
36   /* テクスチャーマッピングを有効にする */
37   glEnable( GL_TEXTURE_2D );
38 }
39
40 void
41 display(void)
42 {
43   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
```

```
44
45   glColor3f( 1.0f, 1.0f, 1.0f );
46
47   /* テクスチャー座標値を与えながら描画 */
48   glBegin( GL_POLYGON );
49   glTexCoord2f( 0.0f, 0.0f );
50   glVertex3f( -35.0f,   0.0f, 0.0f );
51
52   glTexCoord2f( 1.0f, 0.0f );
53   glVertex3f(   0.0f, -35.0f, 0.0f );
54
55   glTexCoord2f( 1.0f, 1.0f );
56   glVertex3f(  35.0f,   0.0f, 0.0f );
57
58   glTexCoord2f( 0.0f, 1.0f );
59   glVertex3f(   0.0f,  35.0f, 0.0f );
60   glEnd();
61
62   glutSwapBuffers();
63 }
64
65 void
66 keyboard( unsigned char c, int x, int y )
67 {
68   if( c==27 ) exit( 0 );
69 }
70
71 void
72 reshape( int w, int h )
73 {
74   glViewport( 0, 0, w, h );
75
76   glMatrixMode( GL_PROJECTION );
77   glLoadIdentity();
78   gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
79   glMatrixMode( GL_MODELVIEW );
80 }
81
82 void
83 animate(void)
84 {
85   glutPostRedisplay();
86 }
87
```

```
 88 int
 89 main( int argc, char *argv[] )
 90 {
 91   glutInit( &argc, argv );
 92
 93   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
 94
 95   glutInitWindowPosition( 100, 100 );
 96   glutInitWindowSize( 500, 400 );
 97   glutCreateWindow( argv[0] );
 98
 99   glutDisplayFunc( display );
100   glutKeyboardFunc( keyboard );
101   glutReshapeFunc( reshape );
102   glutIdleFunc( animate );
103
104   initialize();
105   glutMainLoop();
106   return 0;
107 }
108
```

<演習>

1. forYou ディレクトリの texture プログラムを実行して下さい。

```
% cd forYou
% make texture
% texture
```

2. animation プログラム中の 1 枚ポリゴンの色を白にしてテクスチャーをはるか、白いポリゴンを 1 つ追加してテクスチャーをはって下さい。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

サブルーチンのまとめ

テクスチャーのセットアップ

```
GLvoid glTexImage2D( GLenum target, GLint level, GLint components,
  GLsizei width, GLsizei height, GLint border,
  GLenum format, GLenum type, const GLvoid *image )
```

イメージの拡大縮小

```
GLint gluScaleImage( GLenum format,
  GLsizei widthin, GLint heightin, GLenum typein, const void *datain,
  GLint widthout, GLint heightout, GLenum typeout, void *dataout )
```

テクスチャーのオン/オフ

```
GLvoid glEnable( GL_TEXTURE_2D )
GLvoid glDisable( GL_TEXTURE_2D )
```

テクスチャー座標値の指定

```
GLvoid glTexCoord2fv( const GLfloat *v )
```

第11章　テクスチャーマッピング上級

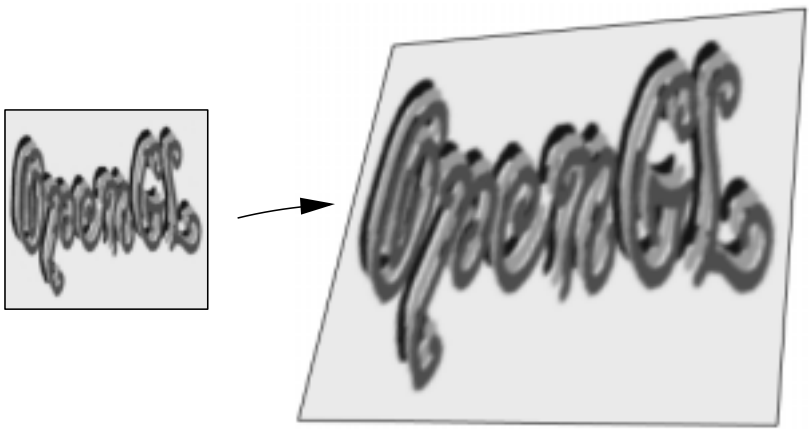
本章の目的

- テクスチャー環境のセットアップを行なう
- 拡大縮小フィルターを理解する
- ミップマップを理解する
- テクスチャー座標値の自動生成方法を習得する
- リフレクションマッピングを行なう
- 高度なテクスチャーマッピングを行なう

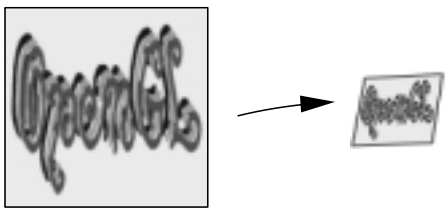
11.1　テクスチャー環境のセットアップ

テクスチャーの拡大縮小フィルター

- テクスチャーポリゴン描画時には、１テクセルが１ピクセルに対応するわけではない
- 拡大フィルター
  – １ピクセルは、１テクセルより小さな領域に対応する
  – テクスチャーは拡大される



- 縮小フィルター
  – １ピクセルは、１テクセルより大きな領域に対応する
  – テクスチャーは縮小される



ミップマップ

- テクスチャーの縮小を単純に行なうと、モデルの動きでテクスチャーがちらつく
- あらかじめ「ミップマップ」と呼ばれる様々なサイズのテクスチャー群を作っておく
- ミップマップをセットアップしておけば、適切なレベルを自動選択する

ミップマップのロード

- ミップマップを自作
  - glTexImage2D() で各レベルごとにロード
  - レベルにより異なるイメージをロードできる
- ミップマップの自動生成
  - gluBuild2DMipmaps()
  - $2^m \times 2^n (m \geq n)$ のイメージでは…
  - gluBuild2DMipmaps() は gluScaleImage() で m 個のミップマップイメージを作り、オリジナルと合わせて m+1 個のミップマップを glTexImage2D() でロードする
  - イメージは、ミップマップ作成前に $2^m \times 2^n$ にしておく

ミップマップの自動生成

GLint gluBuild2DMipmaps(GLenum target, GLint components, GLsizei width, GLint height, GLenum format, GLenum type, void *data)

- target はターゲットテクスチャー
  - 常に GL_TEXTURE_2D
- components はカラーコンポーネント数
  - 1〜4 のいずれか
- width はテクスチャーの幅
- height はテクスチャーの高さ
- format はテクセルフォーマット
  - GL_COLOR_INDEX、GL_RED、GL_GREEN、GL_BLUE、GL_ALPHA、GL_RGB、GL_RGBA、GL_LUMINANCE、GL_LUMINANCE_ALPHA のいずれか
- type はテクセルタイプ
  - GL_UNSIGNED_BYTE、GL_BYTE、GL_BITMAP、GL_UNSIGNED_SHORT、GL_SHORT、GL_UNSIGNED_INT、GL_INT、GL_FLOAT のいずれか
- data はイメージデータ

テクスチャー属性の設定

GLvoid glTexParameterf(GLenum target, GLenum pname, const GLfloat param)

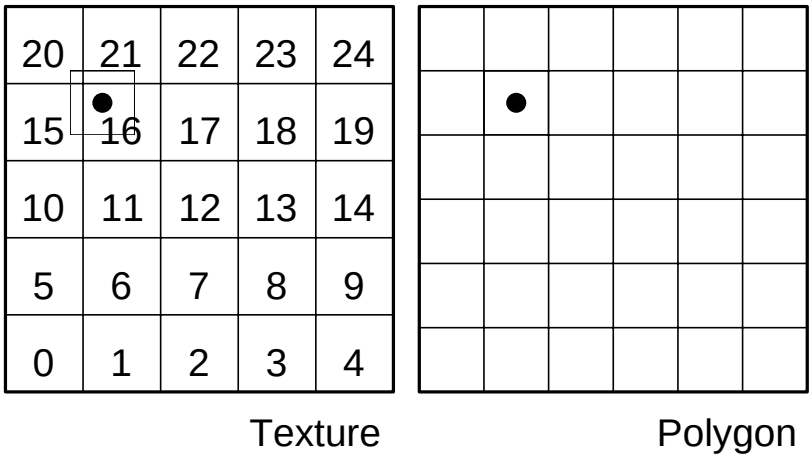
- target はターゲットテクスチャー
  - GL_TEXTURE_1D、GL_TEXTURE_2D のいずれか
- pname はテクスチャープロパティ名
- param はプロパティ値
- テクスチャープロパティに次のものがある

	pname	デフォルト
拡大/縮小フィルター	GL_TEXTURE_MAG_FILTER	GL_LINEAR
	GL_TEXTURE_MIN_FILTER	GL_NEAREST_MIPMAP_NEAREST
範囲外の値の扱い	GL_TEXTURE_WRAP_S	GL_REPEAT
	GL_TEXTURE_WRAP_T	GL_REPEAT
枠の色	GL_TEXTURE_BORDER_COLOR	(0.0, 0.0, 0.0, 0.0)

- 引数指定にスカラー型とベクター型がある

拡大フィルター

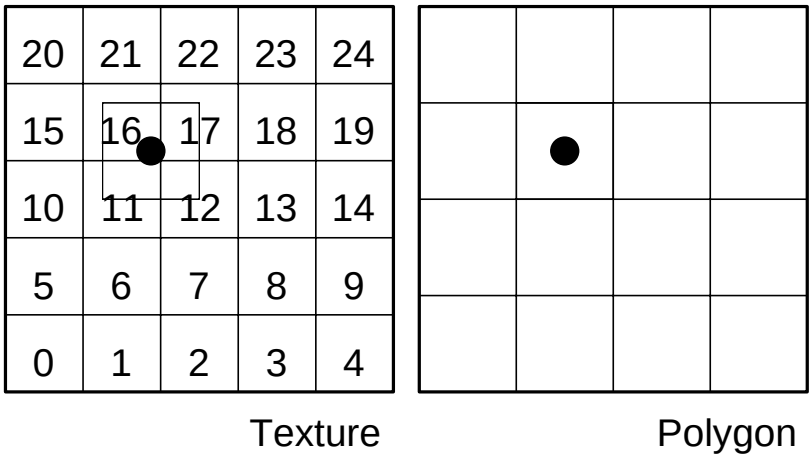
- pname は GL_TEXTURE_MAG_FILTER



- param は GL_NEAREST、GL_LINEAR のいずれか
- GL_NEAREST
  - テクスチャー座標値に最も近いテクセルが使われる
  - 上の例では 16 番テクセル
- GL_LINEAR（デフォルト）
  - テクスチャー座標値に最も近い４テクセルの荷重平均が使われる
  - 上の例では 15、16、20、21 番テクセルの荷重平均

ミップマップを使わない縮小フィルター

- pname は GL_TEXTURE_MIN_FILTER



- param は GL_NEAREST、GL_LINEAR のいずれか
- GL_NEAREST
  - テクスチャー座標値に最も近いテクセルが使われる
  - 上の例では 16 番テクセル
- GL_LINEAR
  - テクスチャー座標値に最も近い 4 テクセルの荷重平均が使われる
  - 上の例では 11、12、16、17 番テクセルの荷重平均

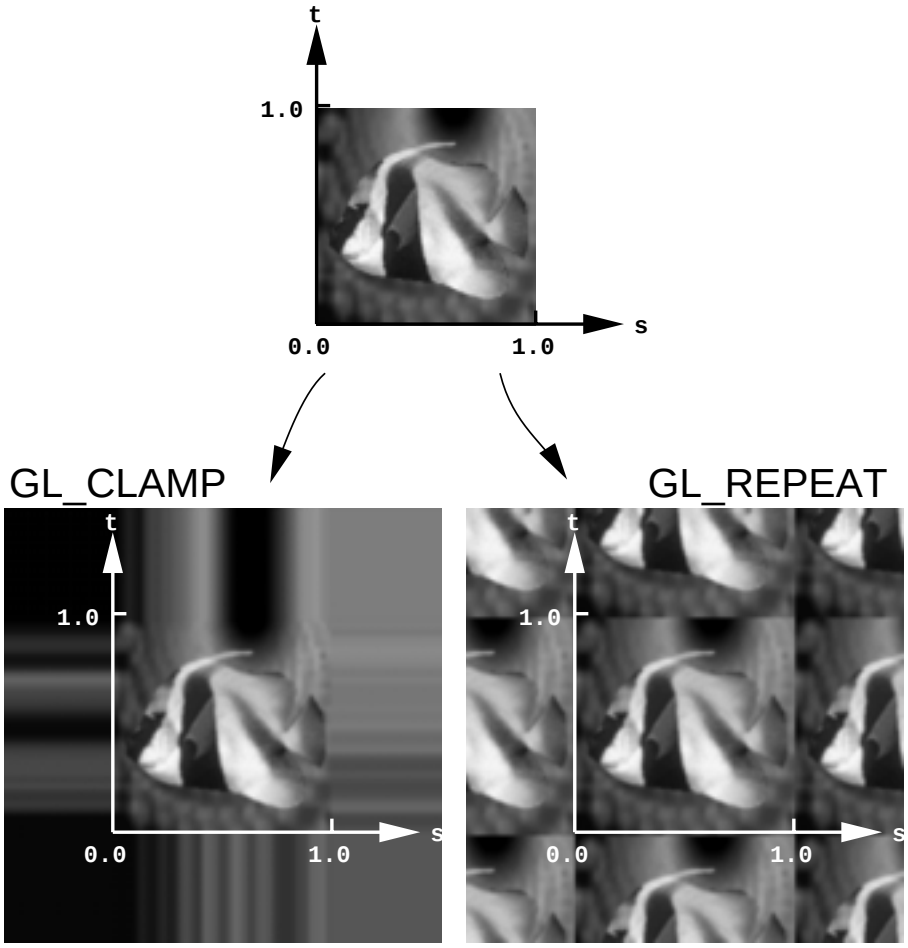
ミップマップを使う縮小フィルター

- pname は GL_TEXTURE_MIN_FILTER
- param は次のいずれか
  - GL_NEAREST_MIPMAP_NEAREST
    大きさの近いミップマップを選択し、テクスチャー座標値に最も近いテクセルが使われる
  - GL_LINEAR_MIPMAP_NEAREST
    大きさの近いミップマップを選択し、テクスチャー座標値に最も近い４テクセルの荷重平均が使われる
  - GL_NEAREST_MIPMAP_LINEAR（デフォルト）
    大きさの近い２つのミップマップを選択し、テクスチャー座標値に最も近いテクセルを荷重平均する
  - GL_LINEAR_MIPMAP_LINEAR
    大きさの近い２つのミップマップを選択し、テクスチャー座標値に最も近い４テクセルの荷重平均を荷重平均する

範囲外のテクスチャー座標値の扱い

テクスチャー座標値は 0.0〜1.0
glTexCoord*() で範囲外のテクスチャー座標値を指定すると…

- pname は GL_TEXTURE_WRAP_S、GL_TEXTURE_WRAP_T のいずれか
- param は GL_CLAMP、GL_REPEAT（デフォルト）のいずれか



オリジナルとテクスチャーとの色の混合環境

GLvoid glTexEnvf(GLenum target, GLenum pname, const GLfloat param)

- ポリゴンのオリジナルカラーとテクスチャーの混合環境を設定
- target は常に GL_TEXTURE_ENV
- pname は テクスチャー環境プロパティ名
- param は プロパティ値
- テクスチャー環境プロパティに次のものがある

	pname	デフォルト
混合モード	GL_TEXTURE_ENV_MODE	GL_MODULATE
参照色	GL_TEXTURE_ENV_COLOR	(0.0, 0.0, 0.0, 0.0)

- 引数指定にスカラー型とベクター型がある

ポリゴンの色はどうなる？

- GL_MODULATE（デフォルト）
  - １コンポーネントテクスチャー：
    $Color_{out} = Color_{in} \times I_{tex}$
    $Alpha_{out} = Alpha_{in}$
  - ２コンポーネントテクスチャー：
    $Color_{out} = Color_{in} \times I_{tex}$
    $Alpha_{out} = Alpha_{in} \times Alpha_{tex}$
  - ３コンポーネントテクスチャー：
    $Color_{out} = Color_{in} \times Color_{tex}$
    $Alpha_{out} = Alpha_{in}$
  - ４コンポーネントテクスチャー：
    $Color_{out} = Color_{in} \times Color_{tex}$
    $Alpha_{out} = Alpha_{in} \times Alpha_{tex}$
- GL_DECAL（３、４コンポーネント用）
  - ３コンポーネントテクスチャー：
    $Color_{out} = Color_{tex}$
    $Alpha_{out} = Alpha_{in}$
  - ４コンポーネントテクスチャー：
    $Color_{out} = Color_{in} \times (1 - Alpha_{tex}) + Color_{tex} \times Alpha_{tex}$
    $Alpha_{out} = Alpha_{in}$
- GL_BLEND（１、２コンポーネント用）
  - １コンポーネントテクスチャー:
    $Color_{out} = Color_{in} \times (1 - I_{tex}) + Color_{const} \times I_{tex}$
    $Alpha_{out} = Alpha_{in}$
  - ２コンポーネントテクスチャー:
    $Color_{out} = Color_{in} \times (1 - I_{tex}) + Color_{const} \times I_{tex}$
    $Alpha_{out} = Alpha_{in} \times Alpha_{tex}$

```
 1 /*
 2 ** texclamp.c - 0-1範囲外のテクスチャー座標値
 3 */
 4
 5 #include <GL/glut.h>
 6 #include "laconia.h"
 7 #include "sgiimage.h"
 8
 9 void
10 initialize(void)
11 {
12   unsigned int* image;
13   GLsizei       width, height;
14
15   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
16
17   glMatrixMode( GL_MODELVIEW );
18   glLoadIdentity();
19   gluLookAt( 0.0, 50.0, 100.0,
20              0.0,  0.0,   0.0,
21              0.0,  1.0,   0.0 );
22
23   glEnable( GL_DEPTH_TEST );
24   glEnable( GL_CULL_FACE );
25
26   image = rgbReadImageFile( "../images/ocean.rgb",
27                             &width, &height );
28
29   /* テクスチャーをミップマップでセットアップ */
30   gluBuild2DMipmaps( GL_TEXTURE_2D, 3, width, height,
31                      GL_RGBA, GL_UNSIGNED_BYTE, image );
32
33   /* S座標についてテクスチャー環境をクランプモードにする */
34   glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
35                    GL_CLAMP );
36
37   glEnable( GL_TEXTURE_2D );
38 }
39
40 void
41 display(void)
42 {
43   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
```

```
44
45   glColor3f( 1.0f, 1.0f, 1.0f );
46
47   glColor3f(1.0f, 1.0f, 1.0f);
48
49   /* 0-1範囲外のテクスチャー座標値を使う */
50   glBegin( GL_POLYGON );
51   glTexCoord2f( 0.0f, 0.0f );
52   glVertex3f( -35.0f,   0.0f, 0.0f );
53   glTexCoord2f( 2.0f, 0.0f );
54   glVertex3f(   0.0f, -35.0f, 0.0f );
55   glTexCoord2f( 2.0f, 2.0f );
56   glVertex3f(  35.0f,   0.0f, 0.0f );
57   glTexCoord2f( 0.0f, 2.0f );
58   glVertex3f(   0.0f,  35.0f, 0.0f );
59   glEnd();
60
61   glutSwapBuffers();
62 }
63
64 void
65 keyboard( unsigned char c, int x, int y )
66 {
67   if( c==27 ) exit( 0 );
68 }
69
70 void
71 reshape( int w, int h )
72 {
73   glViewport( 0, 0, w, h );
74
75   glMatrixMode( GL_PROJECTION );
76   glLoadIdentity();
77   gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
78   glMatrixMode( GL_MODELVIEW );
79 }
80
81 void
82 animate(void)
83 {
84   glutPostRedisplay();
85 }
86
87 int
```

```
 88 main( int argc, char *argv[] )
 89 {
 90   unsigned int* image;
 91   GLsizei       width, height;
 92
 93   glutInit( &argc, argv );
 94
 95   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
 96
 97   glutInitWindowPosition( 100, 100 );
 98   glutInitWindowSize( 500, 400 );
 99   glutCreateWindow( argv[0] );
100
101   glutDisplayFunc( display );
102   glutKeyboardFunc( keyboard );
103   glutReshapeFunc( reshape );
104   glutIdleFunc( animate );
105
106   initialize();
107   glutMainLoop();
108   return 0;
109 }
110
```

<演習>

1. forYou ディレクトリの texclamp プログラムを実行して下さい。

```
% cd forYou
% make texclamp
% texclamp
```

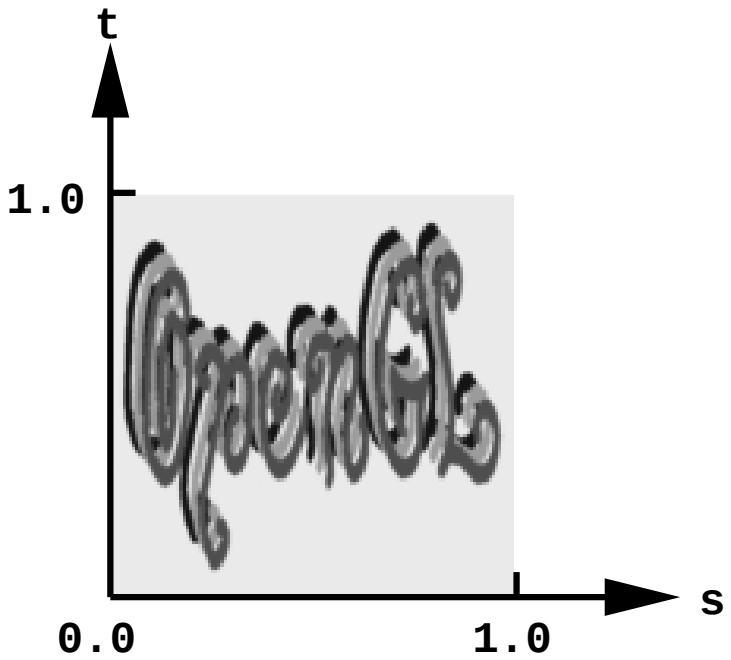
2. animation プログラム中のテクスチャーの属性 (glTexParameterf()) や色の混合環境 (glTexEnvf()) を、適当に変更して下さい。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

11.2 テクスチャー座標値の自動生成

自動生成のために…

- テクスチャー座標値自動生成のセットアップ
- 必要に応じて自動生成用の平面を決める
- 自動生成を行なう
  - s に対して：
    glEnable(GL_TEXTURE_GEN_S);
  - t に対して：
    glEnable(GL_TEXTURE_GEN_T);



自動生成のセットアップ

GLvoid glTexGenfv(GLenum coord, GLenum pname, const GLfloat *params)

- coord はテクスチャー座標軸
  - GL_S、GL_T、GL_R、GL_Q のいずれか
- pname は params で設定する値の名前
  - GL_TEXTURE_GEN_MODE
    自動生成モード
  - GL_OBJECT_PLANE、GL_EYE_PLANE
    自動生成に使う平面
- 引数指定にスカラー型とベクター型がある

自動生成モード

- pname が GL_TEXTURE_GEN_MODE のとき、param は次のいずれか
- GL_OBJECT_LINEAR
  - 座標値をモデル座標系で計算
  - テクスチャーはモデルに固定
  - s、t 用の平面設定が必要
- GL_EYE_LINEAR
  - 座標値をビュー座標系で計算
  - モデルはテクスチャー中を泳ぐ
  - s、t 用の平面設定が必要
- GL_SPHERE_MAP
  - リフレクションマッピング（環境マッピング）
  - 座標値を法線から計算
  - 平面設定は不要

自動生成に使う平面

- pname が GL_OBJECT_PLANE、GL_EYE_PLANE のとき、param は平面方程式の係数配列

$$Ax + By + Cz + D = 0$$

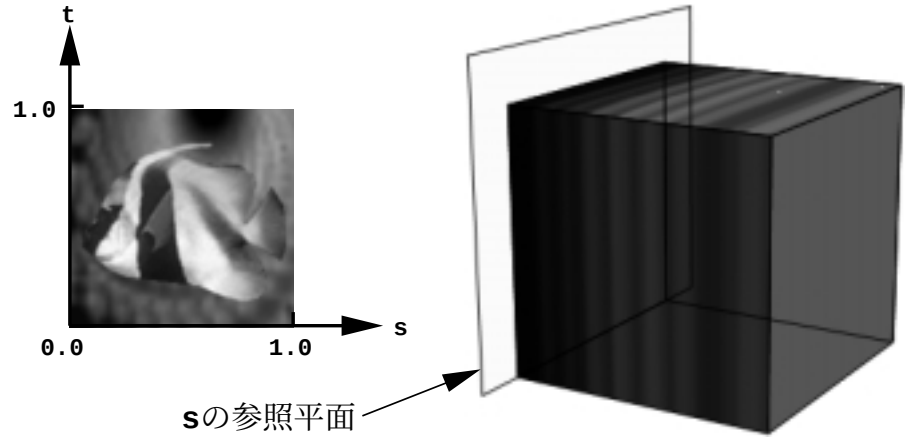
$$param = \{A, B, C, D\}$$

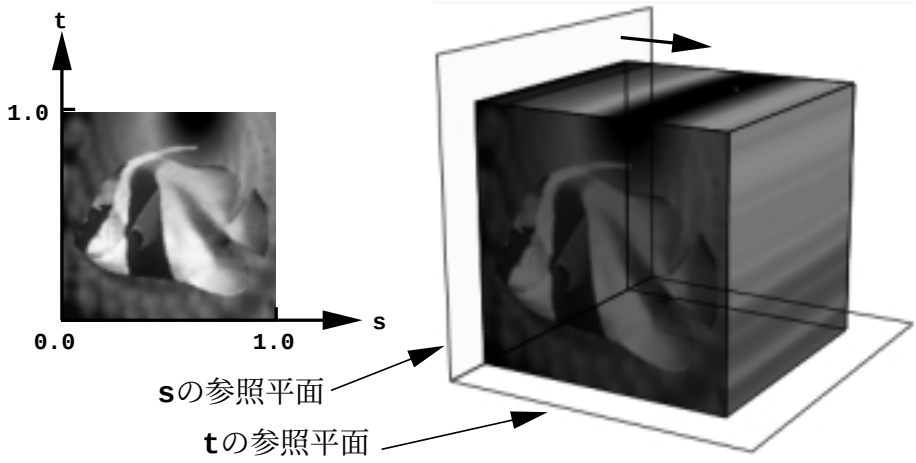
```
GLfloat sPlane = { 1.0, 0.0, 0.0, 0.0 };
GLfloat tPlane = { 0.0, 1.0, 0.0, 0.0 };

glTexGenfv( GL_S, GL_OBJECT_PLANE, sPlane );
glTexGenfv( GL_T, GL_OBJECT_PLANE, tPlane );
```

- GL_OBJECT_LINEAR を使う場合は GL_OBJECT_PLANE を設定
- GL_EYE_LINEAR を使う場合は GL_EYE_PLANE を設定
- GL_SPHERE_MAP を使う場合は 平面設定は不要

GL_OBJECT_LINEAR でのテクスチャー座標値と平面

$$\begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} A_s x + B_s y + C_s z + D_s \\ A_t x + B_t y + C_t z + D_t \end{pmatrix}$$





リフレクションマッピング

- モデルを、周囲の環境を反射しているように描画
- pname に GL_TEXTURE_GEN_MODE、param に GL_SPHERE_MAP を設定
  – GL_SPHERE_MAP は s、t 両方に対して設定する

```
glTexGeni( GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP );
glTexGeni( GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP );
```

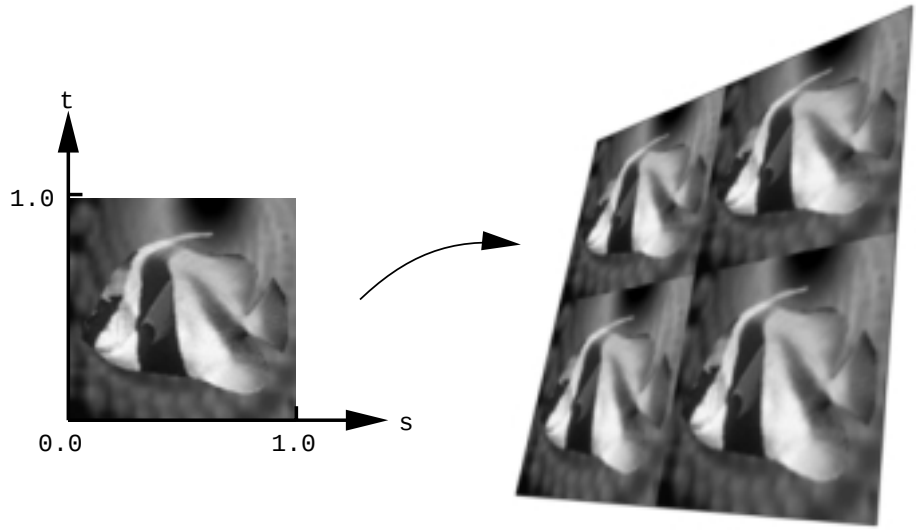
- テクスチャーを、モデルをとりまく球として扱う
- リフレクション用テクスチャーは、魚眼レンズ写真などを使う

11.3 その他

テクスチャーマトリックス

- テクスチャー座標値はモデリング変換可能

```
glMatrixMode( GL_TEXTURE );
glScalef( 2.0, 2.0, 2.0 );
glMatrixMode( GL_MODEL_VIEW );

glBegin( GL_QUADS );
  glTexCoord2f( 0.0, 0.0 );
  glVertex3f( 0.0, 0.0 );
  glTexCoord2f( 1.0, 0.0 );
  glVertex3f( 1.0, 0.0 );
  glTexCoord2f( 1.0, 1.0 );
  glVertex3f( 1.0, 1.0 );
  glTexCoord2f( 0.0, 1.0 );
  glVertex3f( 0.0, 1.0 );
glEnd();
```



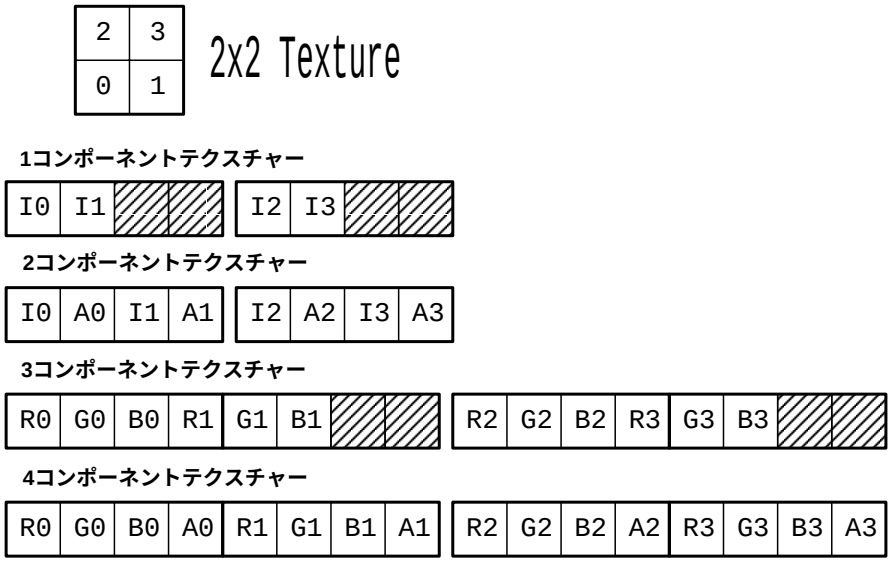
ディスプレイリストによるテクスチャー

- 複数のテクスチャーを使う場合、テクスチャーのセットアップは必ずディスプレイリストにしておく
- 多くのシステム、とりわけテクスチャーメモリを搭載したものでは、ディスプレイリストでのテクスチャーセットアップで性能が飛躍的に向上する

```
GLuint sphTexList;
GLuint cubeTexList;

initTexList()
{
  ...
  sphTexList = glGenLists( 1 );
  glNewList( sphTexList, GL_COMPILE );
    gluBuild2DMipmaps(...);
  glEndList();
  ...
  cubeTexList = glGenLists( 1 );
  glNewList( cubeTexList, GL_COMPILE );
    gluBuild2DMipmaps(...);
  glEndList();
}

draw()
{
  ...
  glCallList( sphTexList );
  drawSph();
  glCallList( cubeTexList );
  drawCube();
  ...
}
```

テクスチャー配列

- テクスチャー配列のフォーマット
  - テクスチャー情報は unsigned int の配列
  - テクセルは左から右、下から上の順に格納される
- 各々のラインは４バイトワード単位
  - ４バイトワード中の使われていないバイトは無視される



```
 1 /*
 2 ** texgen.c - テクスチャー座標値の自動生成
 3 */
 4
 5 #include <GL/glut.h>
 6 #include "laconia.h"
 7 #include "sgiimage.h"
 8
 9 void
10 initialize(void)
11 {
12   unsigned int* image;
13   GLsizei       width, height;
14
15   glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
16
17   glMatrixMode( GL_MODELVIEW );
18   glLoadIdentity();
19   gluLookAt( 0.0, 50.0, 100.0,
20              0.0,  0.0,   0.0,
21              0.0,  1.0,   0.0 );
22
23   glEnable( GL_DEPTH_TEST );
24   glEnable( GL_CULL_FACE );
25
26   /* 魚眼イメージをロードする */
27   image = rgbReadImageFile( "../images/cafe.rgb",
28                             &width, &height );
29   gluBuild2DMipmaps( GL_TEXTURE_2D, 3, width, height,
30                      GL_RGBA, GL_UNSIGNED_BYTE, image );
31   glEnable( GL_TEXTURE_2D );
32
33   /* テクスチャー座標値の自動生成をセットアップする */
34   glTexGeni( GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP );
35   glTexGeni( GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP );
36
37   /* テクスチャー座標値の自動生成を有効にする */
38   glEnable( GL_TEXTURE_GEN_S );
39   glEnable( GL_TEXTURE_GEN_T );
40 }
41
42 void
43 display(void)
```

```
44 {
45   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
46
47   /* テクスチャー座標値は自動生成される */
48   lacMountain( 20.0 );
49
50   glutSwapBuffers();
51 }
52
53 void
54 keyboard( unsigned char c, int x, int y )
55 {
56   if( c==27 ) exit( 0 );
57 }
58
59 void
60 reshape( int w, int h )
61 {
62   glViewport( 0, 0, w, h );
63
64   glMatrixMode( GL_PROJECTION );
65   glLoadIdentity();
66   gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
67   glMatrixMode( GL_MODELVIEW );
68 }
69
70 void
71 animate(void)
72 {
73   glutPostRedisplay();
74 }
75
76 int
77 main( int argc, char *argv[] )
78 {
79   unsigned int* image;
80   GLsizei       width, height;
81
82   glutInit( &argc, argv );
83
84   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
85
86   glutInitWindowPosition( 100, 100 );
```

```
 87   glutInitWindowSize( 500, 400 );
 88   glutCreateWindow( argv[0] );
 89
 90   glutDisplayFunc( display );
 91   glutKeyboardFunc( keyboard );
 92   glutReshapeFunc( reshape );
 93   glutIdleFunc( animate );
 94
 95   initialize();
 96   glutMainLoop();
 97   return 0;
 98 }
 99
```

<演習>

1. forYou ディレクトリの texgen プログラムを実行して下さい。

```
% cd forYou
% make texgen
% texgen
```

2. animation プログラム中のモデルに、テクスチャー座標値自動生成機能を使ってテクスチャーをはって下さい。ただし複数のテクスチャーを使う場合、カレントテクスチャーの変更はディスプレイリストで行なって下さい。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行なう）
% make animation
% animation
```

サブルーチンのまとめ

ミップマップのセットアップ

```
GLint gluBuild2DMipmaps( GLenum target, GLint components,
  GLsizei width, GLint height, GLenum format, GLenum type, void *data )
```

テクスチャー属性の設定

```
GLvoid glTexParameterf( GLenum target, GLenum pname, const GLfloat param )
```

オリジナルとテクスチャーのいろの混合環境の設定

```
GLvoid glTexEnvf( GLenum target, GLenum pname, const GLfloat param )
```

テクスチャー座標値の自動生成のセットアップ

```
GLvoid glTexGenfv( GLenum coord, GLenum pname, const GLfloat *params )
```

自動生成のオン/オフ

```
GLvoid glEnable( GL_TEXTURE_GEN_S )
GLvoid glDisable( GL_TEXTURE_GEN_S )
GLvoid glEnable( GL_TEXTURE_GEN_T )
GLvoid glDisable( GL_TEXTURE_GEN_T )
```

第12章　テキスト出力

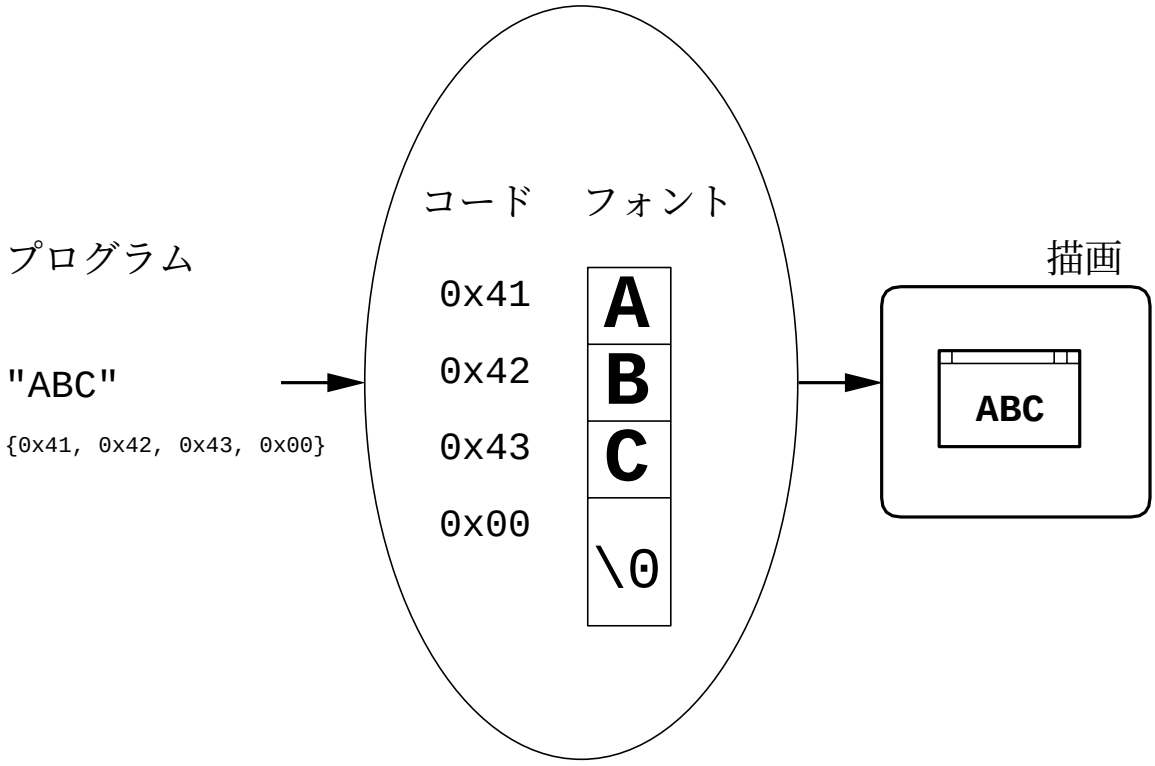
本章の目的

- コードとフォント
- ビットマップフォントとストロークフォント
- ディスプレイリストによるテキスト出力
- GLUT によるテキスト出力

12.1　コードとフォント

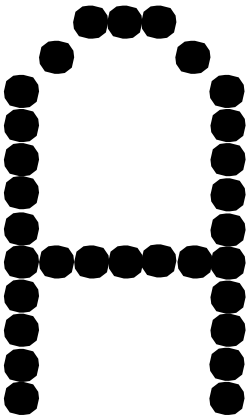
コードとフォントとは

- C のプログラムなどでは、”ABC” と文字列を記述する
  - ここで、’A’、’B’、’C’ はそれぞれ、ASCII コードで 0x41, 0x42, 0x43 に対応する
  - さらに終端を表す、0x00 コードを最後に加え合計 4byte になる
  - この文字列の終端を 0x00 を付加して表現することを、NULL ターミネートするという
- そこで、コンピュータは文字列”ABC”を描画するとき、0x41 番目に A というフォント、0x42 番目に B というフォント、0x43 番目に C というフォントを割り当てて、ABC を描画する
  - 最後に 0x00 を検出したところで、文字列描画の処理を終える
- コードは各文字の識別番号である
- フォントは各文字を描画する際に必要な形状データの集まりである



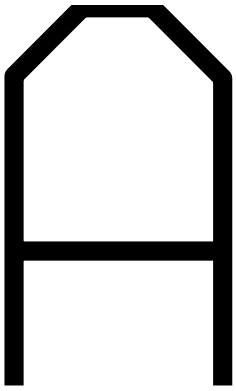
フォントの種類

- ビットマップフォントは文字形状を「点の集まり」で表現する



ビットマップフォント

- ストロークフォントは文字形状を「線の組み合わせ」で表現する



ストロークフォント

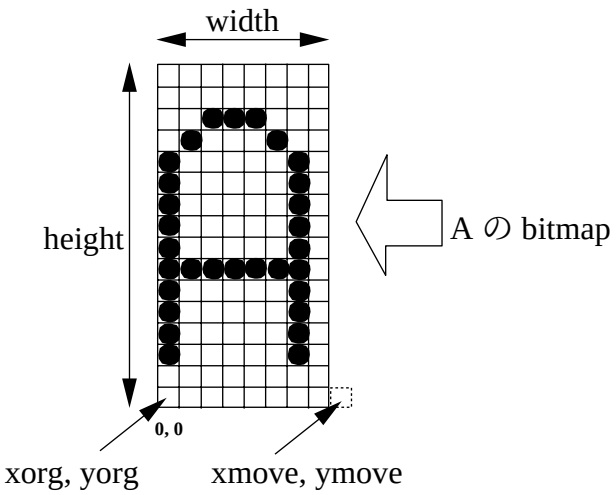
- アウトラインフォントは文字形状を「文字の輪郭情報の組み合わせ」で表現する
- 文字の描画は一般的にウィンドウシステムに依存する
- 本コースではプラットホームに依存しない部分を、ビットマップフォントとストロークフォントを使って説明する

12.2　ディスプレイリストによるテキスト出力

ビットマップフォントとディスプレイリスト

- ディスプレイリスト一つに、一つの文字データを登録して文字/文字列の描画を行う
- ディスプレイリストの i 番目に bitmap を登録する

```
Glubyte bitmap[] = { ... };
glNewList(i, GL_COMPILE);
glBitmap(width, height, xorig, yorig, xmove, ymove, bitmap);
glEndList();
```



  - width は、ビットマップデータの幅
  - height は、ビットマップデータの高さ
  - xorg, yorg は、ビットマップの原点
  - xmove, ymove は、次のビットマップを描画するときの原点
- ディスプレイリストの i 番目を描画する

```
glCallList(i);
```

- glCallList では一文字しか描画できない
  - glCallLists を使うことにより、複数の文字、すなわち文字列を出力できる
  - glCallLists は予め文字セットをディスプレイリストに登録して利用する
    (次ページ以降に解説)

フォントセットのディスプレイリスト化

- ビットマップのフォントセットをディスプレイリストの配列として登録する
  – glGenLists で文字数分のディスプレイリスト番号を取得する
  – ループを使い glBitmap で各文字をディスプレイリスト化する

- プログラムの実装例

```
/*
 * ディスプレイリスト領域を 128(0x80) 個獲得
 */
 base = glGenLists(128);

/*
 * 各文字のビットマップデータを各ディスプレイリストに割り当てる
 */
 for (n = 0, i = 32; i < 128; i++) {
     glNewList(base + i, GL_COMPILE);
     glBitmap(8, 16, 0.0, 0.0, 9.0, 0.0, font[n++]);
     glEndList();
 }
```

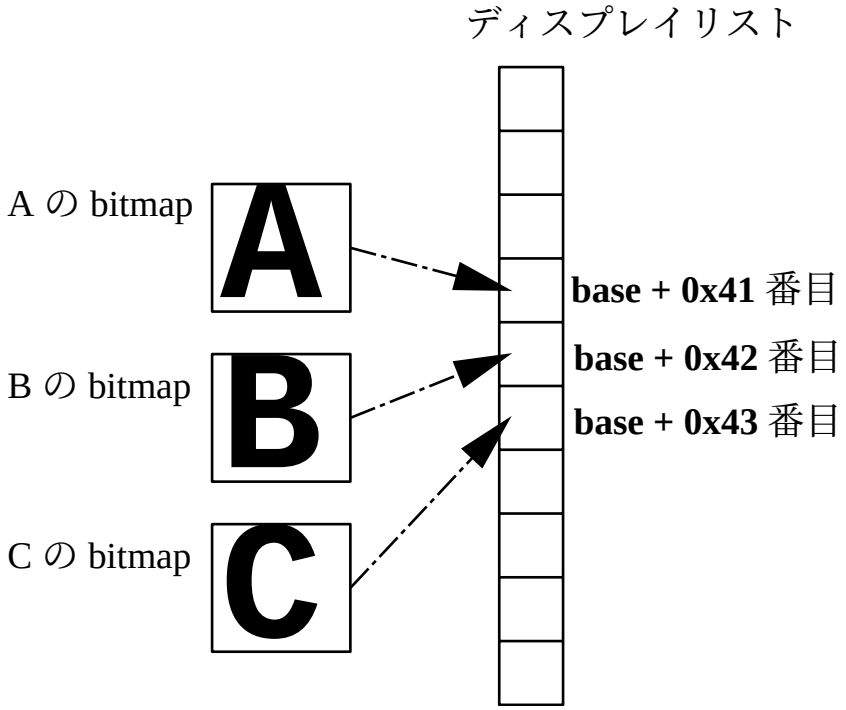
  – ASCII コードの 0 番から 31 番までは特殊コードで形状を持たない
  – font[] 配列に、 8 X 16 のフォントが入っている
  – 各フォントの原点は (0.0, 0.0) とする
  – 次のフォントの描画位置 (ラスターポジション) は (9.0, 0.0) 移動した地点とする

- 登録した文字の描画

```
glListBase(base);
glCallLists(strlen(string), GL_UNSIGNED_BYTE, (GLubyte *)string);
```

フォントセットのディスプレイリスト化 (図解)

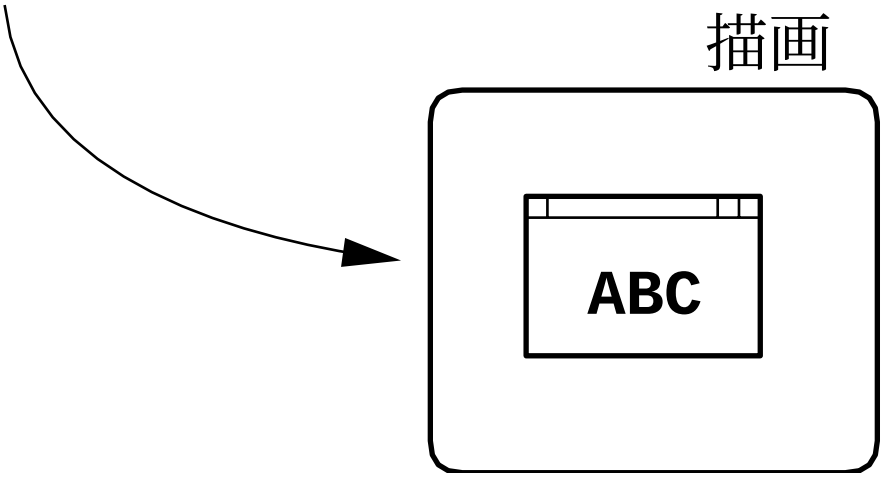
- ビットマップフォントのディスプレイリスト化を図解すると次のようになる



strcpy(**string**, "ABC");

```
string = {0x41, 0x42, 0x43, 0x00}
```

glListBase(**base**);
glCallLists(strlen(string), GL_UNSINGED_BYTE, (GLubyte *)**string**);



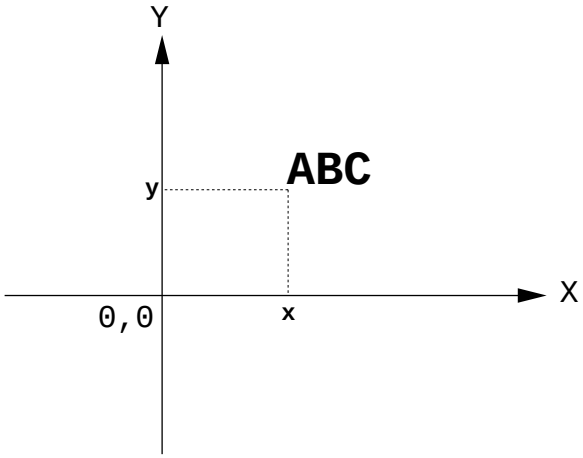
ビットマップフォントの文字列描画

- 文字列を描画する方法

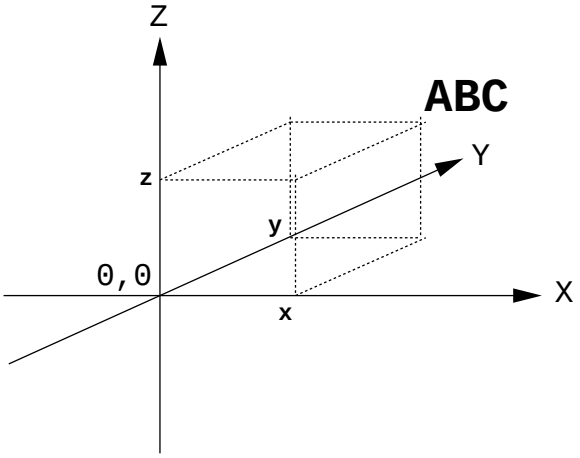
```
strcpy( string, "ABC" );
glListBase(base);
glCallLists(strlen(string), GL_UNSIGNED_BYTE, (GLubyte *)string);
```

glListBase() で、ディスプレイリスト領域を獲得したときの base を設定する

- 文字列を描画する位置
  - ビットマップフォントはラスタープリミティブなので描画位置の制御は glRasterPos* 関数で行う
  - glRasterPos2f(x, y);



  - glRasterPos3f(x, y, z);

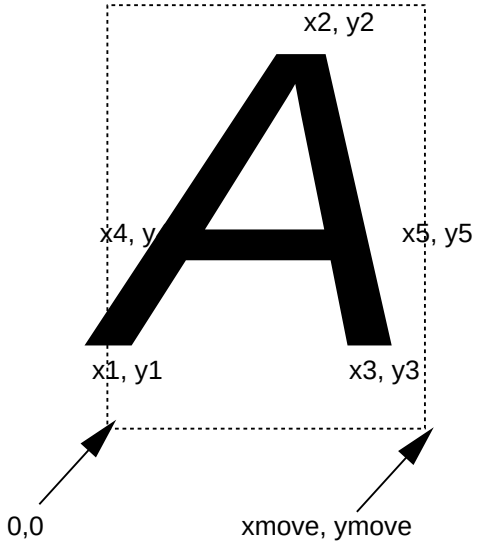


- ビットマップフォントの文字列は、三次元空間でも二次元空間でも描画できるが、書出し位置のみが各種座標変換の影響を受ける
  - フォントの向き、大きさなどは固定である

ストロークフォントとディスプレイリスト

- ストロークフォントはジオメトリープリミティブなので、位置、形状ともモデリング変換の影響を受ける
  - ラスタープリミティブでないため glRasterPos*の影響は受けない
- ストロークフォントの文字列書出し位置はモデリング座標系の原点となる
  - glTranslatef を使って設定する
- ストロークフォントの設定方法
  - ディスプレイリストの i 番目に線の組み合わせで文字を登録する
  - 文字を書き終わったら、glTranslatef で次の文字の書出し位置まで、モデリング座標系を移動させる
  - コーディング例

```
glNewList(i, GL_COMPILE)

    /* i 番目の文字を先で描画 */
    glBegin(GL_LINE_...)
    glVertex2f(x1, y1)
    ...
    glEnd()

    /* 次の文字の書出し位置に移動 */
    glTranslatef(xmove, ymove, 0.0)
    glEndList()
```



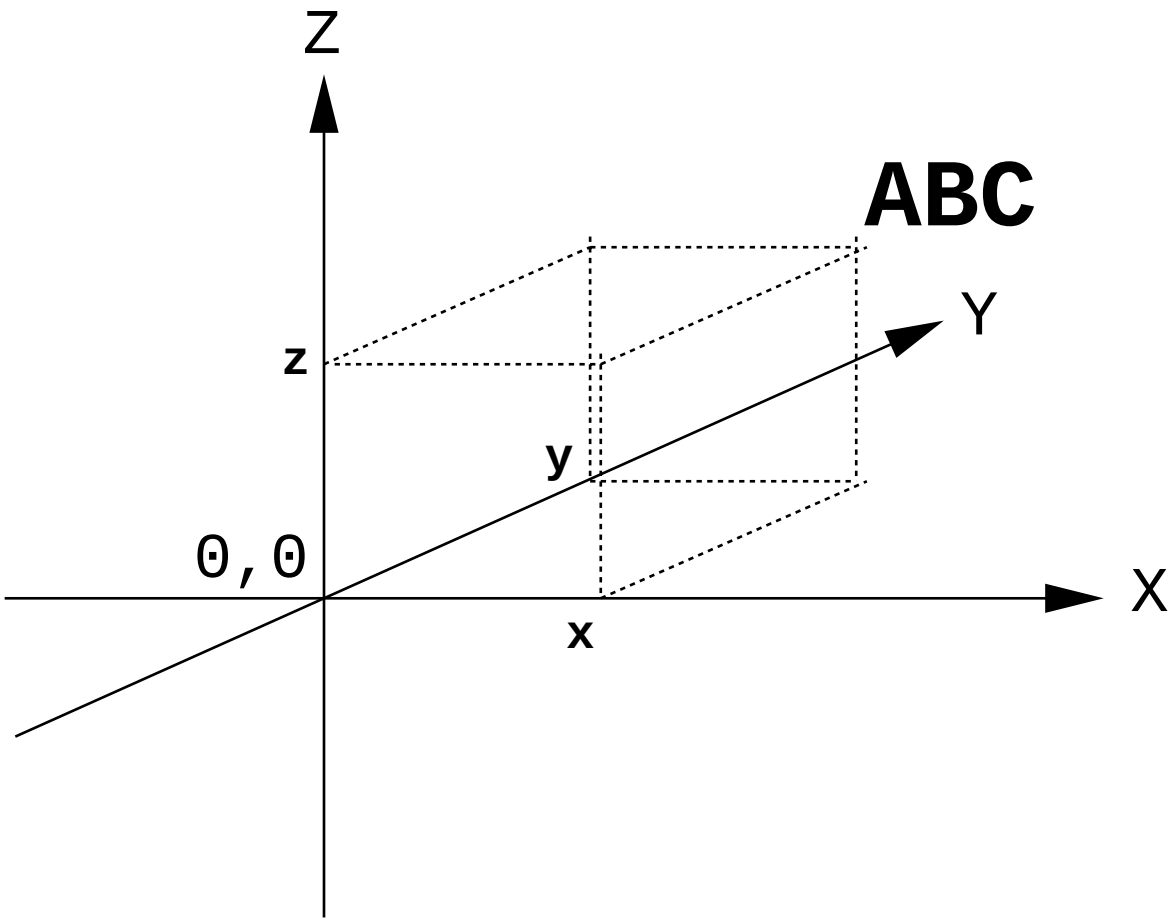
ストロークフォントの文字列描画

- 文字列を描画する方法

```
strcpy( string, "ABC" );
glListBase(base);
glCallLists(strlen(string), GL_UNSIGNED_BYTE, (GLubyte *)string);
```

- 文字列を描画する位置

```
glTranslatef(x, y, z);
```



12.3　GLUT によるテキスト出力

GLUT ビットマップフォントの使い方

- GLUT ではディスプレイリストを使わずにビットマップフォントの文字列出力ができる
  – フォントの初期化処理はなにもしない

- font と文字を指定して glutBitmapCharacter 関数を呼ぶ

```
void glutBitmapCharacter(void *font, int character);
```

- font で指定できるビットマップフォントは次の通り
  – GLUT_BITMAP_8_BY_13
    8 x 13 pixel の固定幅フォント
  – GLUT_BITMAP_9_BY_15
    9 x 15 pixel の固定幅フォント
  – GLUT_BITMAP_TIMES_ROMAN_BY_10
    10 ポイント、プロポーショナルの Times Roman フォント
  – GLUT_BITMAP_TIMES_ROMAN_24
    24 ポイント、プロポーショナルの Times Roman フォント
  – GLUT_BITMAP_HELVETICA_10
    10 ポイント、プロポーショナルの Helvetica フォント
  – GLUT_BITMAP_HELVETICA_12
    12 ポイント、プロポーショナルの Helvetica フォント
  – GLUT_BITMAP_HELVETICA_18
    18 ポイント、プロポーショナルの Helvetica フォント

- 文字を描画する位置はカレントのモデリング空間中の座標値で glRasterPos*で指定する

```
void glRasterPos2f(x, y);
void glRasterPos3f(x, y, z);
```

- 文字の幅 (画面上の pixel 数) を調べるには glutBitmapWidth 関数を使う

```
int glutBitmapWidth(void *font, int character);
```

ビットマップフォントによる文字列の出力

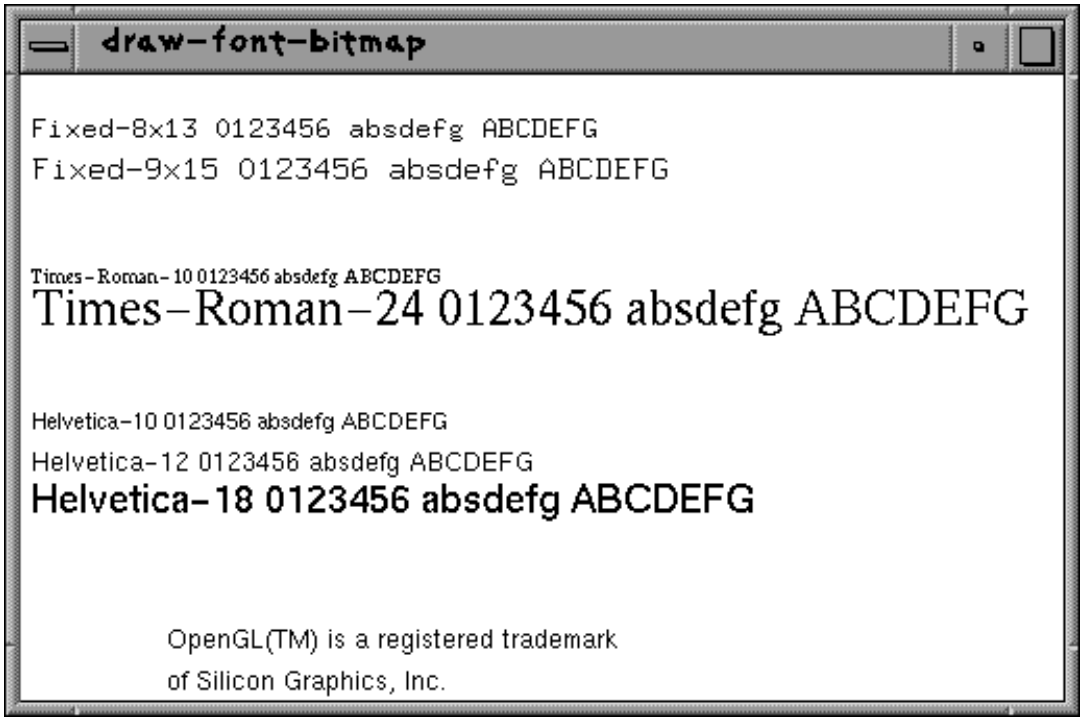
- glutBitmapCharacter は一文字の出力の後に、カレントのモデリング座標系を一文字分移動させる

- 文字列の出力は、文字列の書き出し位置を glRasterPos*で指定し、続いて glutBimapCharacter を連続的に呼び出せばよい

- コーディング例

```
void
drawBitmapString(int x, int y, void *font, char *string)
{
   glRasterPos2f(x, y);
   while (*string)
       glutBitmapCharacter(font, *string++);
}
```

- 出力例



GLUT ストロークフォントの使い方

- GLUT ではディスプレイリストを使わずにストロークフォントの文字列出力ができる
  – フォントの初期化処理はなにもしない

- font と文字を指定して glutStrokeCharacter 関数を呼ぶ

```
void glutStrokeCharacter(void *font, int character);
```

- font で指定できるストロークフォントは次の 2 種類
  – GLUT_STROKE_ROMAN
    プロポーショナルの Roman フォント
  – GLUT_STROKE_MONO_ROMAN 固定スペースの Roman フォント

- 文字を描画する位置はカレントのモデリング空間の原点を基準とする
  描画位置の移動は glPushMatrix/glPopMatrix と組み合わせて、glTranslate* で行う

```
void glTranslatef(GLfloat x, GLfloat y, GLfloat z);
void glTranslated(GLdoublex x, GLdouble y, GLdouble z);
```

- 文字の幅 (画面上の pixel 数) を調べるには glutStrokeWidth 関数を使う

```
int glutStrokeWidth(void *font, int character);
```

ストロークフォントによる文字列の出力

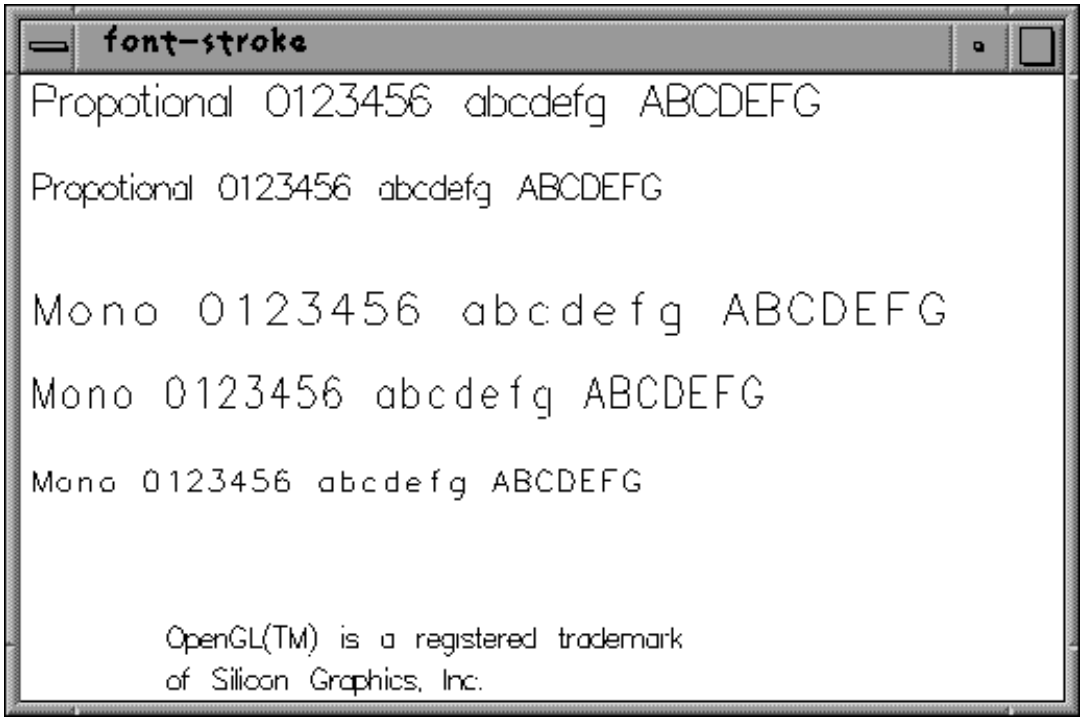
- glutStrokeCharacter は一文字の出力の後に、カレントのモデリング座標系を一文字分移動させる

- ストロークフォントの文字列の出力方法
  - 必ず glPushMatrix と glPopMatrix で囲む
  - glTranslate*で文字列の書き出し位置を指定
  - glScale*で文字の大きさを指定
  - glutStrokeCharacter を連続的に呼び出す

- コーディング例

```
void
drawStrokeString(GLfloat x, GLfloat y, float xscl, float yscl,
                 void *font, char *string)
{
    glPushMatrix();
    glTranslatef(x, y, 0);
    glScalef(xscl, yscl, 1.0f);
    while (*string)
        glutStrokeCharacter(font, *string++);
    glPopMatrix();
}
```

- 出力例



その他の文字の描き方

- GLC という OpenGL 専用のポータブルな文字出力のライブラリを整備するという動きもある
  – '97 年前半現在、SGI 専用でアルファベットのみ
  – 国際化とヨーロッパ系、アジア系文字へのローカリゼーションが課題

- 文字出力は一般にウィンドウシステムと密接に結び付いたものである

- GLX、WGL などウィンドウシステムとのインターフェイスライブラリが文字出力の機能を持っている

- アウトラインフォントの利用は WGL などを使う
  – Windows では WGL を使って TrueType フォントによる漢字出力が可能

- X Window System の場合には Xlib との混合プログラミング環境で、各種フォントを自由に使うことができる
  – OepnGL の座標と、X の座標の整合性をアプリケーションが管理する必要がある

```
 1 /*
 2  * font.c
 3  *
 4  * 配列で用意したビットマップデータによる文字列表示のサンプル
 5  *
 6  */
 7 #include <GL/glut.h>
 8 #include "font.h"
 9
10 GLuint base;
11
12 void
13 initialize()
14 {
15     int i, n;
16
17     glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
18
19    /*
20     * ディスプレイリスト領域を 128 個獲得
21     */
22     base = glGenLists(0x80);
23
24    /*
25     * 各文字のビットマップデータを各ディスプレイリストに割り当てる
26     */
27     n = 0;
28     for (i = 0x20; i < 0x80; i++) {
29  glNewList(base + i, GL_COMPILE);
30  glBitmap(8, 16, 0.0f, 0.0f, 9.0f, 0.0f, font[n++]);
31  glEndList();
32     }
33
34     glClearColor(1.0f, 1.0f, 1.0f, 0.0f);
35 }
36
37 void
38 drawString(char *string)
39 {
40     glPushAttrib(GL_LIST_BIT);
41
42    /*
43     * 文字フォントディスプレイリストの開始位置
44     */
```

```
45      glListBase(base);
46
47     /*
48      * ディスプレイリスト（文字列）を描画
49      */
50      glCallLists(strlen(string), GL_UNSIGNED_BYTE, (GLubyte *)string);
51
52      glPopAttrib();
53  }
54
55  void
56  display(void)
57  {
58      glClear(GL_COLOR_BUFFER_BIT);
59      glColor3f(0.0f, 0.0f, 0.0f);
60
61      glRasterPos2f(5.0f, 280.0f); /* 描画位置 */
62      drawString(" !\"#$%&'()*+,-./0123456789:;<=>?"); /* 文字列描画 */
63
64      glRasterPos2f(5.0f, 260.0f); /* 描画位置 */
65      drawString("@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_"); /* 文字列描画 */
66
67      glRasterPos2f(5.0f, 240.0f); /* 描画位置 */
68      drawString("`abcdefghijklmnopqrstuvwxyz{|}~"); /* 文字列描画 */
69
70      glRasterPos2f(70.0f, 25.0f); /* 描画位置 */
71      drawString("OpenGL(TM) is a trademark"); /* 文字列描画 */
72      glRasterPos2f(70.0f, 5.0f); /* 描画位置 */
73      drawString("of Silicon Graphics, Inc."); /* 文字列描画 */
74
75      glFlush();
76  }
77
78  void
79  keyboard(unsigned char key, int x, int y)
80  {
81      if (key == 0x1B) exit(0);
82  }
83
84  void
85  reshape(int width, int height)
86  {
87      glViewport(0, 0, width, height);
88      glLoadIdentity();
```

```
 89     glOrtho(0.0, (GLdouble)width, 0.0, (GLdouble)height, 0.0, 1.0);
 90 }
 91
 92 int
 93 main(int argc, char* argv[])
 94 {
 95     glutInit( &argc, argv );
 96     glutInitDisplayMode( GLUT_SINGLE | GLUT_RGB );
 97     glutInitWindowPosition( 100, 100 );
 98     glutInitWindowSize( 400, 300 );
 99     glutCreateWindow( argv[0] );
100     glutReshapeFunc( reshape );
101     glutDisplayFunc( display );
102     glutKeyboardFunc( keyboard );
103
104     initialize();
105     glutMainLoop();
106     return 0;
107 }
108
```

```
 1 /*
 2  * font-bitmap.c GLUT のビットマップフォントを使うサンプル
 3  */
 4 #include <GL/glut.h>
 5 #include "font.h"
 6
 7 void
 8 initialize(void)
 9 {
10     glClearColor(1.0f, 1.0f, 1.0f, 0.0f);
11 }
12
13 void
14 drawBitmapString(GLfloat x, GLfloat y, void *font, char *string)
15 {
16     /* 現在のラスターポジションの保存 */
17     glPushAttrib(GL_CURRENT_BIT);
18
19     glRasterPos2f(x, y);
20
21     /* ビットマップ文字列の描画 */
22     while (*string)
23  glutBitmapCharacter(font, *string++);
24
25     /* 保存したラスターポジションのロード */
26     glPopAttrib();
27 }
28
29 void
30 display(void)
31 {
32     glClear(GL_COLOR_BUFFER_BIT);
33     glColor3f(0.0f, 0.0f, 0.0f);
34
35     drawBitmapString(5.0f, 270.0f,
36       GLUT_BITMAP_8_BY_13,
37       "Fixed-8x13 0123456 absdefg ABCDEFG");
38
39     drawBitmapString(5.0f, 250.0f,
40       GLUT_BITMAP_9_BY_15,
41       "Fixed-9x15 0123456 absdefg ABCDEFG");
42
43     drawBitmapString(5.0f, 200.0f,
44       GLUT_BITMAP_TIMES_ROMAN_10,
```

```
45        "Times-Roman-10 0123456 absdefg ABCDEFG");
46
47      drawBitmapString(5.0f, 180.0f,
48        GLUT_BITMAP_TIMES_ROMAN_24,
49        "Times-Roman-24 0123456 absdefg ABCDEFG");
50
51      drawBitmapString(5.0f, 130.0f,
52        GLUT_BITMAP_HELVETICA_10,
53        "Helvetica-10 0123456 absdefg ABCDEFG");
54
55      drawBitmapString(5.0f, 110.0f,
56        GLUT_BITMAP_HELVETICA_12,
57        "Helvetica-12 0123456 absdefg ABCDEFG");
58
59      drawBitmapString(5.0f,  90.0f,
60        GLUT_BITMAP_HELVETICA_18,
61        "Helvetica-18 0123456 absdefg ABCDEFG");
62
63      drawBitmapString(70.0f, 25.0f,
64        GLUT_BITMAP_HELVETICA_12,
65        "OpenGL(TM) is a registered trademark");
66      drawBitmapString(70.0f,  5.0f,
67        GLUT_BITMAP_HELVETICA_12,
68        "of Silicon Graphics, Inc.");
69
70      glFlush();
71  }
72
73  void
74  keyboard(unsigned char key, int x, int y)
75  {
76      const unsigned char ESCAPE_KEY = 0x1B;
77      if (key == ESCAPE_KEY) exit(0);
78  }
79
80  void
81  reshape(int width, int height)
82  {
83      glViewport(0, 0, width, height);
84      glLoadIdentity();
85      glOrtho(0.0, (GLdouble)width, 0.0, (GLdouble)height, 0.0, 1.0);
86  }
87
88  int
```

```
 89 main(int argc, char* argv[])
 90 {
 91     glutInit( &argc, argv );
 92     glutInitDisplayMode( GLUT_SINGLE | GLUT_RGB );
 93     glutInitWindowPosition( 100, 100 );
 94     glutInitWindowSize( 500, 300 );
 95     glutCreateWindow( argv[0] );
 96
 97     glutReshapeFunc( reshape );
 98     glutDisplayFunc( display );
 99     glutKeyboardFunc( keyboard );
100
101     initialize();
102     glutMainLoop();
103     return 0;
104 }
105
```

```
 1 /*
 2  * font-stroke.c GLUT のストロークフォントを使うサンプル
 3  */
 4 #include <GL/glut.h>
 5 #include "font.h"
 6
 7 void
 8 initialize(void)
 9 {
10     glClearColor(1.0f, 1.0f, 1.0f, 0.0f);
11 }
12
13 void
14 drawStrokeString(GLfloat x, GLfloat y, float xscl, float yscl,
15   void *font, char *string)
16 {
17     glPushMatrix();
18     glTranslatef(x, y, 0.0f);
19     glScalef(xscl, yscl, 1.0f);
20
21     /* ストロークフォント文字列の描画 */
22     while (*string)
23  glutStrokeCharacter(font, *string++);
24
25     glPopMatrix();
26 }
27
28 void
29 display(void)
30 {
31     glClear(GL_COLOR_BUFFER_BIT);
32     glColor3f(0.0f, 0.0f, 0.0f);
33
34     drawStrokeString(5.0f, 280.0f, 0.15f, 0.15f,
35       GLUT_STROKE_ROMAN,
36       "Propotional 0123456 abcdefg ABCDEFG");
37
38     drawStrokeString(5.0f, 240.0f, 0.12f, 0.12f,
39       GLUT_STROKE_ROMAN,
40       "Propotional 0123456 abcdefg ABCDEFG");
41
42     drawStrokeString(5.0f, 180.0f, 0.15f, 0.15f,
43       GLUT_STROKE_MONO_ROMAN,
44       "Mono 0123456 abcdefg ABCDEFG");
```

```
45
46      drawStrokeString(5.0f, 140.0f, 0.12f, 0.15f,
47        GLUT_STROKE_MONO_ROMAN,
48        "Mono 0123456 abcdefg ABCDEFG");
49
50      drawStrokeString(5.0f, 100.0f, 0.10f, 0.10f,
51        GLUT_STROKE_MONO_ROMAN,
52        "Mono 0123456 abcdefg ABCDEFG");
53
54      drawStrokeString(70.0f, 25.0f, 0.1f, 0.1f,
55        GLUT_STROKE_ROMAN,
56        "OpenGL(TM) is a registered trademark");
57      drawStrokeString(70.0f,  5.0f, 0.1f, 0.1f,
58        GLUT_STROKE_ROMAN,
59        "of Silicon Graphics, Inc.");
60
61      glFlush();
62  }
63
64  void
65  keyboard(unsigned char key, int x, int y)
66  {
67      const unsigned char ESCAPE_KEY = 0x1B;
68      if (key == ESCAPE_KEY) exit(0);
69  }
70
71  void
72  reshape(int width, int height)
73  {
74      glViewport(0, 0, width, height);
75      glLoadIdentity();
76      glOrtho(0.0, (GLdouble)width, 0.0, (GLdouble)height, 0.0, 1.0);
77  }
78
79  int
80  main(int argc, char* argv[])
81  {
82      glutInitDisplayMode( GLUT_SINGLE | GLUT_RGB );
83      glutInitWindowPosition( 100, 100 );
84      glutInitWindowSize( 500, 300 );
85      glutCreateWindow( argv[0] );
86
87      glutReshapeFunc( reshape );
88      glutDisplayFunc( display );
```

```
89      glutKeyboardFunc( keyboard );
90
91      initialize();
92      glutMainLoop();
93      return 0;
94  }
95
```

<演習>

1. ビットマップフォントをディスプレイリストを使って描画したサンプルを確認してください。

```
% vi font.c
% make font
% font
```

2. GLUT による bitmap フォントを使ったテキスト出力のサンプルを確認してください。

```
% vi font-bitmap.c
% make font-bitmap
% font-bitmap
```

3. GLUT による stroke フォントを使ったテキスト出力のサンプルを確認してください。

```
% vi font-stroke.c
% make font-stroke
% font-stroke
```

4. animation プログラムにテキストの出力を追加してください。

```
% cd bobby
% vi animation.c
...（適当にプログラムの変更を行う）
% make animation
% animation
```

サブルーチンのまとめ

ディスプレイリスト

```
void glNewList( GLuint list, GLenum mode )
void glEndList( void )
void glCallList( GLuint list )
void glCallLists( GLsizei n, GLenum type, const GLvoid *lists )
GLuint glGenLists( GLsizei range )
void glListBase( GLuint base )
```

ビットマップの描画

```
void glBitmap( GLsizei width, GLsizei height,
               GLfloat xorig, GLfloat yorig,
               GLfloat xmove, GLfloat ymove,
               const GLubyte *bitmap )
```

ピクセルオペレーションのためのラスタ位置の指定

```
void glRasterPos2f( GLfloat x, GLfloat y )
void glRasterPos3f( GLfloat x, GLfloat y, GLfloat z )
```

GLUT 関数

```
void glutBitmapCharacter(void *font, int character);
int  glutBitmapWidth(void *font, int character);
void glutStrokeCharacter(void *font, int character);
int  glutStrokeWidth(void *font, int character);
```

第13章　イメージデータ

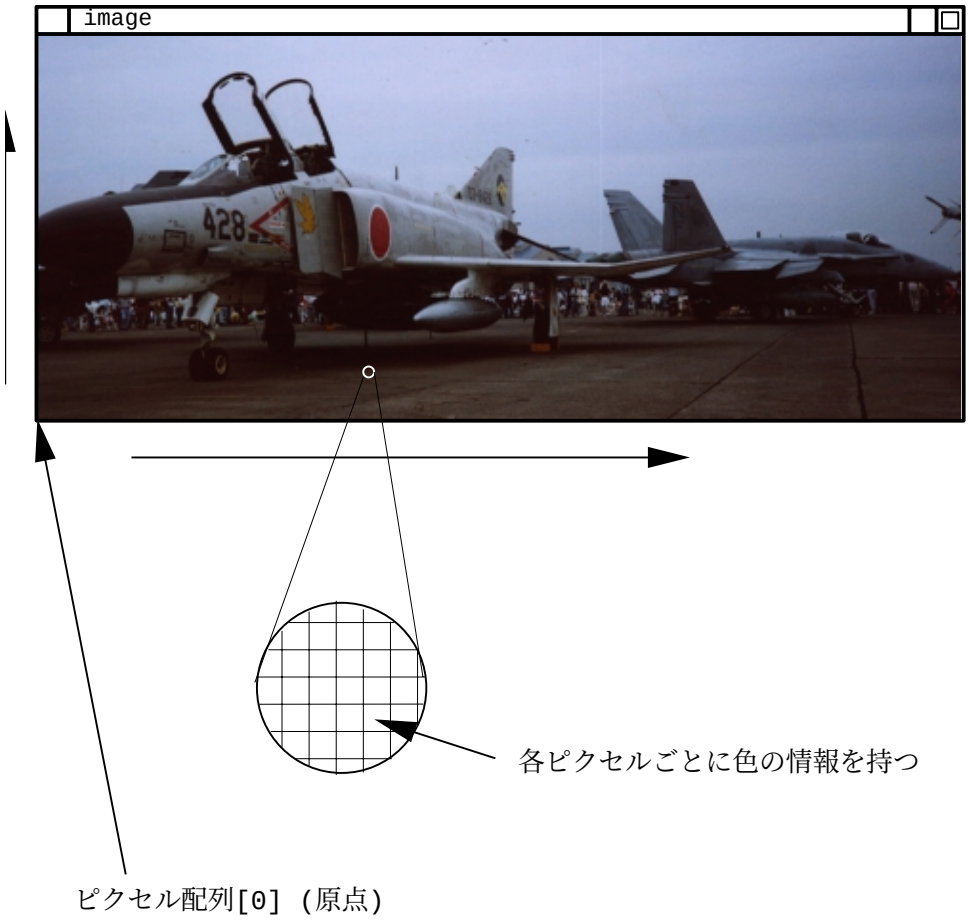
本章の目的

- イメージデータの読み込み
- イメージファイルの描画
- イメージファイルの書き込み
- イメージのフレームバッファ内でのコピー

13.1　イメージデータの取り扱い

イメージデータはラスタープリミティブ

- OpenGL ではイメージデータはビットマップフォントと同様にラスタープリミティブとして取り扱う
- 原点のワールド空間中の描画位置は glRasterPos*関数で移動する
  - イメージの左下が原点となる
  - glRasterPos*関数で指定する描画位置はモデリング変換の影響を受ける
    しかし位置が変るだけでイメージの描画自体は変換の影響を受けない
- イメージの拡大縮小は gluScaleImage 関数を使う
  - glScale*関数はイメージの大きさに影響を与えない



イメージファイルの読み込みと描画

- OpenGL Version 1.0/1.1 自体には特定の外部イメージファイル形式はなく、イメージファイルをロードする機能はない
  - GLUT Version 3.4 以降では、フリーウェアの libtiff を使った TIFF ファイルをロードする例題が含まれる
  - AUX ライブラリには SGI の RGB ファイルローダーが含まれるが、アルファ値を取り扱えない
  - Windows 版 AUX ライブラリには Windows BMP ファイルのローダーが含まれる
  - ImageVision などのライブラリを使う方法も可能

- 本講習では SGI の RGB ファイルを読み込むユーティリティライブラリを使う
  このライブラリはアルファ値を扱える

```
unsigned int *rgbReadImageFile(char *name, int *pWidth, int *pHeight);
```

  - name は SGI イメージのファイル名
  - pWidth, pHeight ポインタ変数の差すアドレスに、イメージの幅と高さを返す
  - 返り値は、unsigne int のイメージ配列の先頭アドレス
    読み込みに失敗したときは NULL が返る

- メインメモリのイメージ配列データをフレームバッファに書き込んで描画する

```
void glDrawPixels(GLsizei width, GLsizei height,
                  GLenum format, GLenum type, const GLvoid *pixels );
```

  - width はイメージの幅, height はイメージの高さ (単位はピクセル)
  - format はイメージ配列の形式である
    例えばアルファ値が不要で RGB だけ必要な場合は GL_RGB を、
    アルファ値も含めたい場合は GL_RGBA を指定する
    rgbReadImageFile を使って読み込んだ場合は、常に GL_RGBA を指定
  - type は配列の各要素の型名である
    例えば、GL_UNSIGNED_BYTE を指定する
  - image はイメージ配列の先頭アドレス

OpenGL ウィンドウのイメージを配列に読み込む

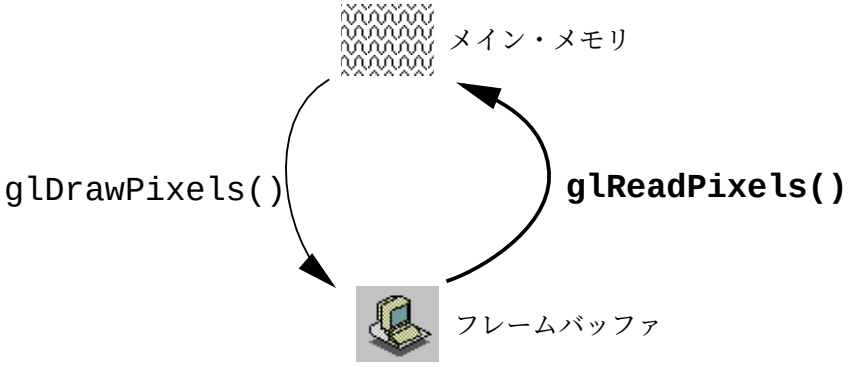
- フレームバッファからピクセルデータを配列に読み込む

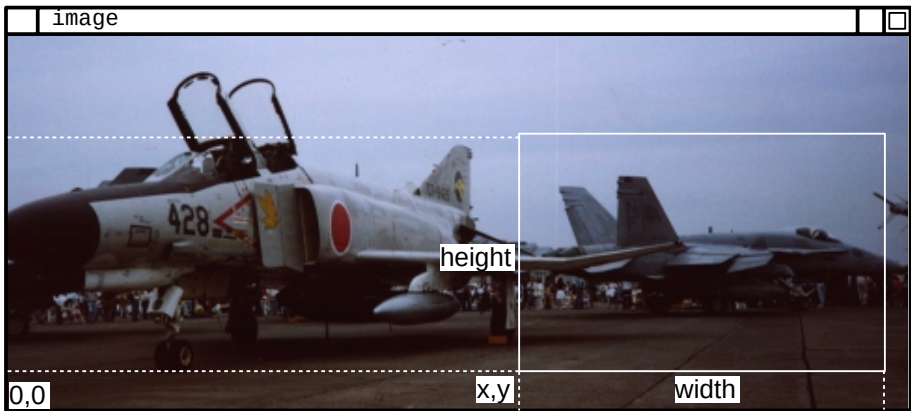
```
void glReadPixels(GLint x, GLint y, GLsizei width, GLsizei height,
                  GLenum format, GLenum type, GLvoid *pixels)
```

  - pixels に、あらかじめ必要なメモリ領域を確保する

```
unsigned int *pixels;
pixels = (unsigned int *)malloc(width*height*sizeof(int))
```

  - もとのイメージの x, y 位置を原点にして、幅 width ピクセル、高さ height ピクセルが、pixels 配列に保存される
  - format はイメージ配列の形式で、通常アルファ値が必要なければ GL_RGB を、アルファ値が必要なら GL_RGBA を指定する後述の rgbWriteImageFile を使って書き込む場合は、常に GL_RGBA を指定
  - type は配列の各要素の型名である
    例えば、GL_UNSIGNED_BYTE を指定する





イメージ配列を外部ファイルに保存する

- OpenGL Version 1.0/1.1 では特定の外部イメージファイル形式を定義していない
  そのためイメージファイルに書き込む機能はない
  – GLUT Version 3.4 以降では、フリーウェアの libtiff を使って、tiff 形式ファイルに書き込む例題がある
  – AUX ライブラリにも SGI の RGB ファイルローダーは含まれるが、ファイルにセーブする機能はない

- 本講習では SGI の RGB ファイルを書き込むユーティリティライブラリを使う

```
int rgbWriteImageFile
        (char *name, int width, int height, unsigned int *parray);
```

  – name はファイル名
  – width、height は、それぞれイメージの幅と高さ
  – parray は rgba イメージ配列の先頭アドレス
  – 書き込みが正常終了したときは 1 を、失敗したときは 0 を返す

フレームバッファのイメージデータをコピーする

- フレームバッファのある領域のイメージデータを、
  別の領域にコピーすることができる

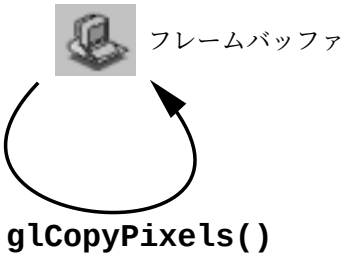
- はじめにコピー先の原点座標 (newx, nwey) を glRasterPos*関数で指定する

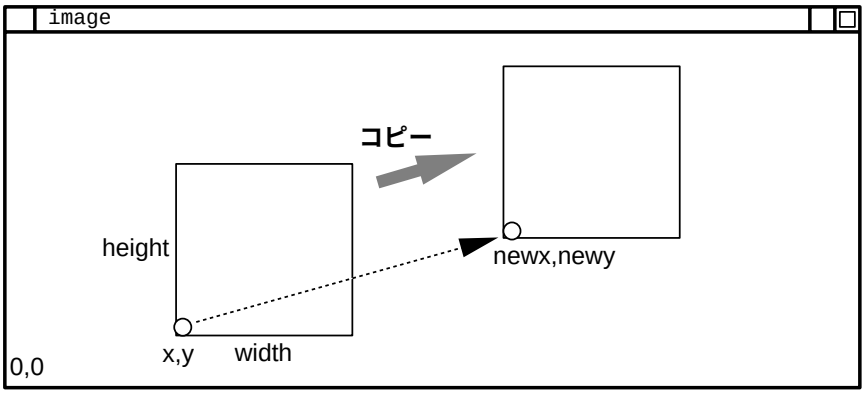
```
glRasterPos2f(newx, newy);
```

- (newx, newy) に、glCopyPixels でイメージデータをコピーする

```
void glCopyPixels(GLint x, GLint y,
                  GLsizei width, GLsizei height, GLenum type)
```

  – x, y はコピーもとの原点
  – 幅 width ピクセル、高さ height ピクセルをコピーして描画する
  – type はコピー元のフレームバッファの種類を指定する
    GL_COLOR, GL_DEPTH, または GL_STENCIL
  – ピクセルデータのコピーは次の 3 種類の関数でコントロールできる
    glPixelTransfer, glPixelMap, glPixelZoom





```
 1 /*
 2  * drawimage.c
 3  *
 4  * fstimage ライブラリを使った SGI イメージファイル表示のサンプル
 5  * (Windows でも動作可能)
 6  *
 7 */
 8 #include <GL/glut.h>
 9 #include <stdio.h>
10 #include "sgiimage.h"
11
12 static int xsize, ysize;
13 static unsigned int *image;
14
15 void
16 loadimage(char *imagefile)
17 {
18     /* SGI イメージファイルからデータを配列にロード */
19     image = rgbReadImageFile(imagefile, &xsize, &ysize);
20     if (!image)
21     {
22  fprintf(stderr, "Unable to open SGI image file: %s\n", imagefile);
23  exit(1);
24     }
25 }
26
27 void
28 initialize(void)
29 {
30     /* イメージの各行は byte アラインメント */
31     glPixelStorei(GL_UNPACK_ALIGNMENT,1);
32 }
33
34 void
35 display(void)
36 {
37     /* ラスターポジションを設定 */
38     glRasterPos2i(0,  0);
39
40     /* イメージデータを表示する */
41     glDrawPixels(xsize, ysize, GL_RGBA, GL_UNSIGNED_BYTE, image);
42     glFlush();
43 }
44
```

```
45 void
46 reshape(int width, int height)
47 {
48     glViewport(0, 0, width, height);
49
50     glMatrixMode (GL_PROJECTION);
51     glLoadIdentity();
52
53     /* 画面のピクセルと 1 対 1 対応するようにクリッピングウィンドウを設定 */
54     glOrtho(0.0, ((double)width-1.), 0.0, ((double)height-1.), -1.0, 1.0);
55     glMatrixMode (GL_MODELVIEW);
56 }
57
58 void
59 keyboard(unsigned char key, int x, int y)
60 {
61     const unsigned char ESCAPE_KEY = 0x1B;
62     if (key == ESCAPE_KEY) exit(0);
63 }
64
65 int
66 main(int argc, char **argv)
67 {
68     glutInit( &argc, argv);
69
70     if (argc != 2)
71     {
72         fprintf(stderr,"Usage: %s <sgi-image-file>\n", argv[0]);
73         exit(1);
74     }
75     loadimage(argv[1]);
76
77     glutInitDisplayMode( GLUT_RGB | GLUT_SINGLE );
78     glutInitWindowPosition( 100, 100 );
79     glutInitWindowSize( xsize, ysize );
80     glutCreateWindow( argv[0] );
81
82     glutReshapeFunc( reshape );
83     glutDisplayFunc( display );
84     glutKeyboardFunc( keyboard );
85
86     initialize();
87     glutMainLoop();
88     return 0;
```

```
89 }
90
```

```
 1 /*
 2 ** saveimage.c - SGI イメージファイルへの保存
 3 */
 4
 5 #include <stdio.h>
 6 #include <GL/glut.h>
 7 #include "laconia.h"
 8 #include "sgiimage.h"
 9
10 static char *imagefile;
11
12 void
13 initialize(void)
14 {
15     glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
16
17     glEnable( GL_DEPTH_TEST );
18     glEnable( GL_CULL_FACE );
19
20     glEnable( GL_LIGHTING );
21     glEnable( GL_LIGHT0 );
22     glEnable( GL_NORMALIZE );
23
24     glMatrixMode(GL_MODELVIEW);
25     glLoadIdentity();
26     gluLookAt( 0.0, 50.0, 100.0,
27                0.0,  0.0,   0.0,
28         0.0,  1.0,   0.0 );
29 }
30
31 void
32 saveimage(void)
33 {
34     int width  = glutGet(GLUT_WINDOW_WIDTH);
35     int height = glutGet(GLUT_WINDOW_HEIGHT);
36
37     /* イメージデータのためのメモリー領域を確保 */
38     unsigned int *pixels
39  = (unsigned int *)malloc(width*height*sizeof(unsigned int));
40
41     /* フレームバッファーからイメージを読み込む */
42     glReadPixels(0, 0, width, height, GL_RGBA, GL_UNSIGNED_BYTE, pixels);
43
44     /* イメージを外部ファイルの保存する */
```

```
45     if (rgbWriteImageFile(imagefile, width, height, pixels))
46  fprintf(stderr, "Saved: %s\n", imagefile);
47     else
48  fprintf(stderr, "Unable to save: %s\n", imagefile);
49 }
50
51 void
52 keyboard( unsigned char c, int x, int y )
53 {
54     /* ESC を押すと終了、"p"キーを押すとイメージファイルのセーブ */
55     if (c == 27) exit(0);
56     else if (c == 'P' || c == 'p') saveimage();
57 }
58
59 void
60 display(void)
61 {
62     static GLfloat ad[] = { 0.0, 1.0, 0.0, 1.0 };
63     static GLfloat shininess[] = { 50.0 };
64
65     glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
66
67     /* 色指定の代わりにマテリアルプロパティを設定する */
68     glMaterialfv( GL_FRONT, GL_AMBIENT_AND_DIFFUSE, ad );
69     glMaterialfv( GL_FRONT, GL_SHININESS, shininess );
70
71     lacMountain( 20.0 );
72     glutSwapBuffers();
73 }
74
75 void
76 reshape( int w, int h )
77 {
78     glViewport( 0, 0, w, h );
79
80     glMatrixMode( GL_PROJECTION );
81     glLoadIdentity();
82     gluPerspective( 45.0, 1.0, 1.0, 1000.0 );
83     glMatrixMode( GL_MODELVIEW );
84 }
85
86 void
87 animate(void)
88 {
```

```
 89     glutPostRedisplay();
 90 }
 91
 92 int
 93 main( int argc, char *argv[] )
 94 {
 95     glutInit( &argc, argv );
 96
 97     if (argc != 2)
 98     {
 99  fprintf(stderr, "Usage: %s <image-file-name>\n", argv[0]);
100  return 1;
101     }
102     imagefile = argv[1];
103
104     glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
105
106     glutInitWindowPosition( 100, 100 );
107     glutInitWindowSize( 500, 400 );
108
109     /* 使用方法をウィンドウタイトルに表示 */
110     glutCreateWindow( "Press p key to save the image" );
111
112     glutDisplayFunc( display );
113     glutKeyboardFunc( keyboard );
114     glutReshapeFunc( reshape );
115     glutIdleFunc( animate );
116
117     initialize();
118     glutMainLoop();
119     return 0;
120 }
121
```

<演習>

1. forYou ディレクトリの saveimage プログラム、darwimage プログラムを実行してください。
   はじめに saveimage プログラムで p キーを押してイメージファイルを生成します。
   続いて drawimage プログラムで、そのイメージファイルを表示します。

```
% make saveimage
% saveimage new.rgb
% drawimage new.rgb
```

2. 各自の animation プログラムに saveimage プログラムと同様に、p キーを押すとイメージファイルを保存する機能を加えて見ましょう。
   saveimage.c を参考にしてください。saveimage 関数はそのまま使えます。

サブルーチンのまとめ

SGI イメージファイル操作関数

```
unsigned int *rgbReadImageFile(char *name, int *pWidth, int *pHeight);

int rgbWriteImageFile
         (char *name, int width, int height, unsigned int *parray);
```

フレームバッファの読み書き

```
void glDrawPixels( GLsizei width, GLsizei height,
                   GLenum format, GLenum type,
                   const GLvoid *pixels );

void glReadPixels( GLint x, GLint y,
                   GLsizei width, GLsizei height,
                   GLenum format, GLenum type, GLvoid *pixels );
```

フレームバッファのピクセルのコピー

```
void glCopyPixels( GLint x, GLint y,
                   GLsizei width, GLsizei height, GLenum type );
```

ピクセル演算用のラスタ位置の指定

```
void glRasterPos2f( GLfloat x, GLfloat y );
```

付 録A　X/Motif との混合プログラミング (1) 基礎編

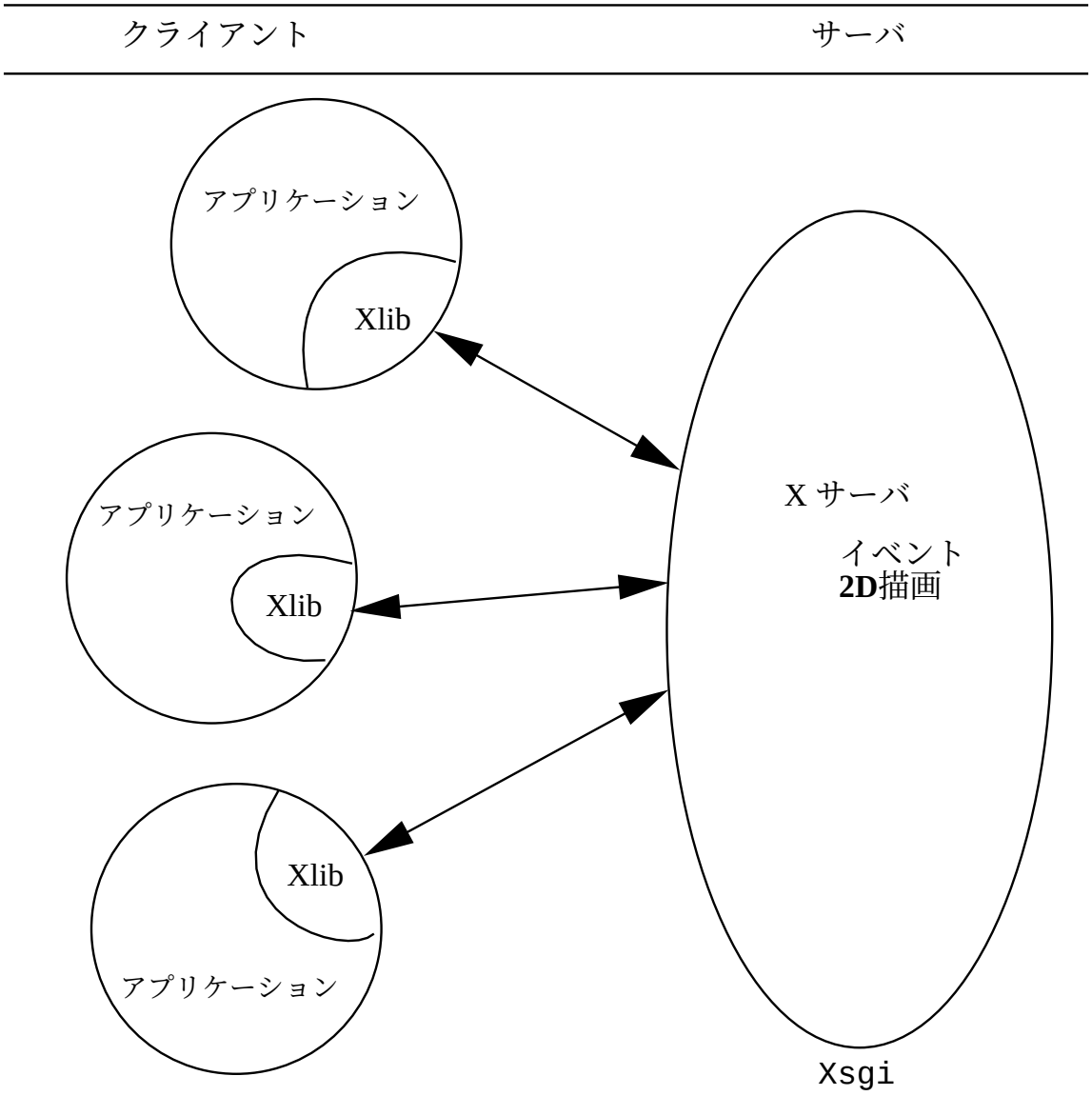
本章の目的

- X Window System と OpenGL の関係
- X Window 用の OpenGL 拡張コマンド (GLX)
- X Toolkit と Motif の概要
- Motif と OpenGL の混合プログラミング
- アニメーション
- 入力イベント

A.1　X Window System と OpenGL の関係

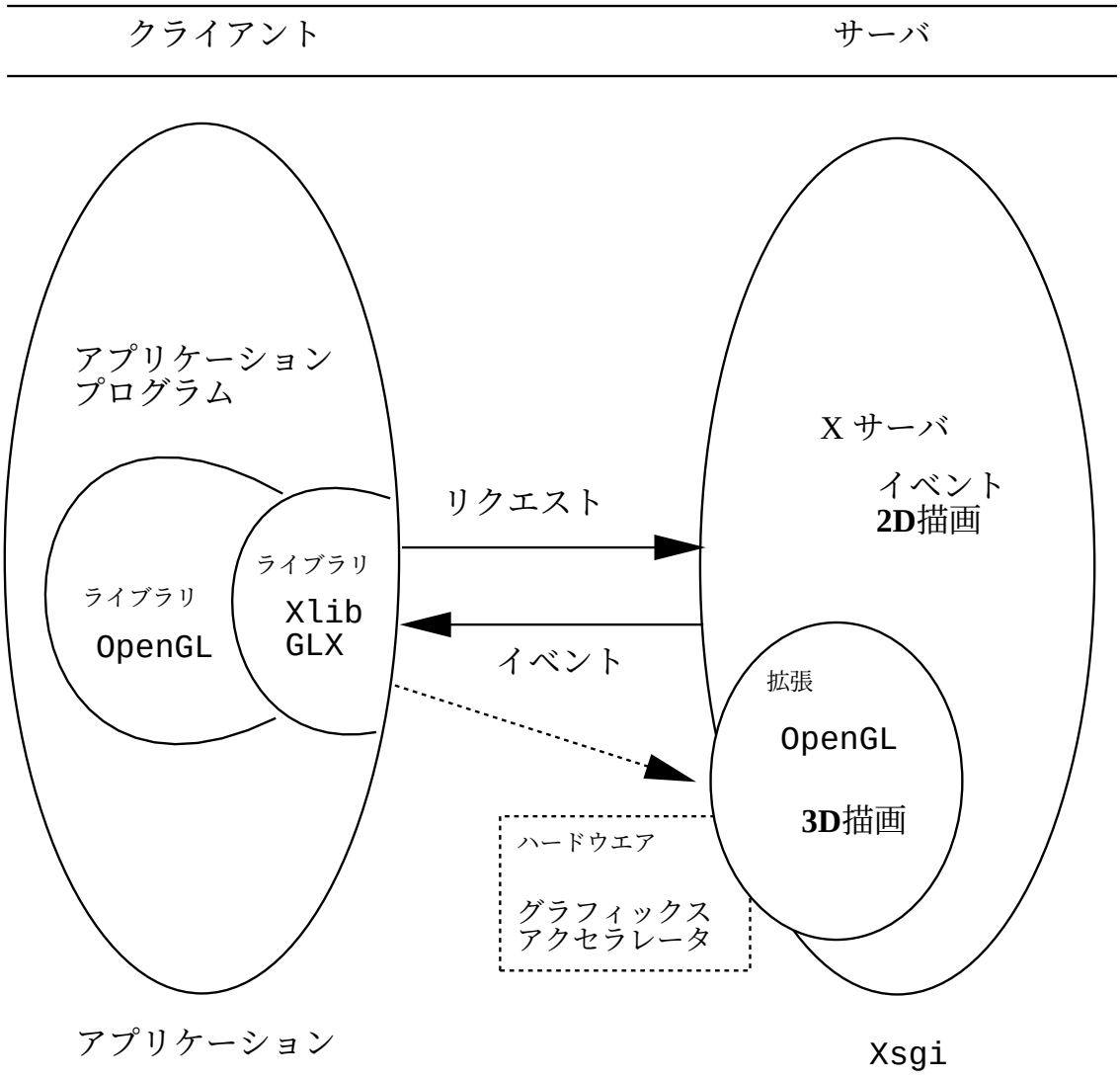
基本的な X Window System

- X Window System のアプリケーションは、すべて Xlib を使う
- X Window System は、サーバ/クライアント
- X Window System のサーバは、1 つのディスプレイに 1 つ動いている
- Xlib は、X Window System の C 言語インターフェィス



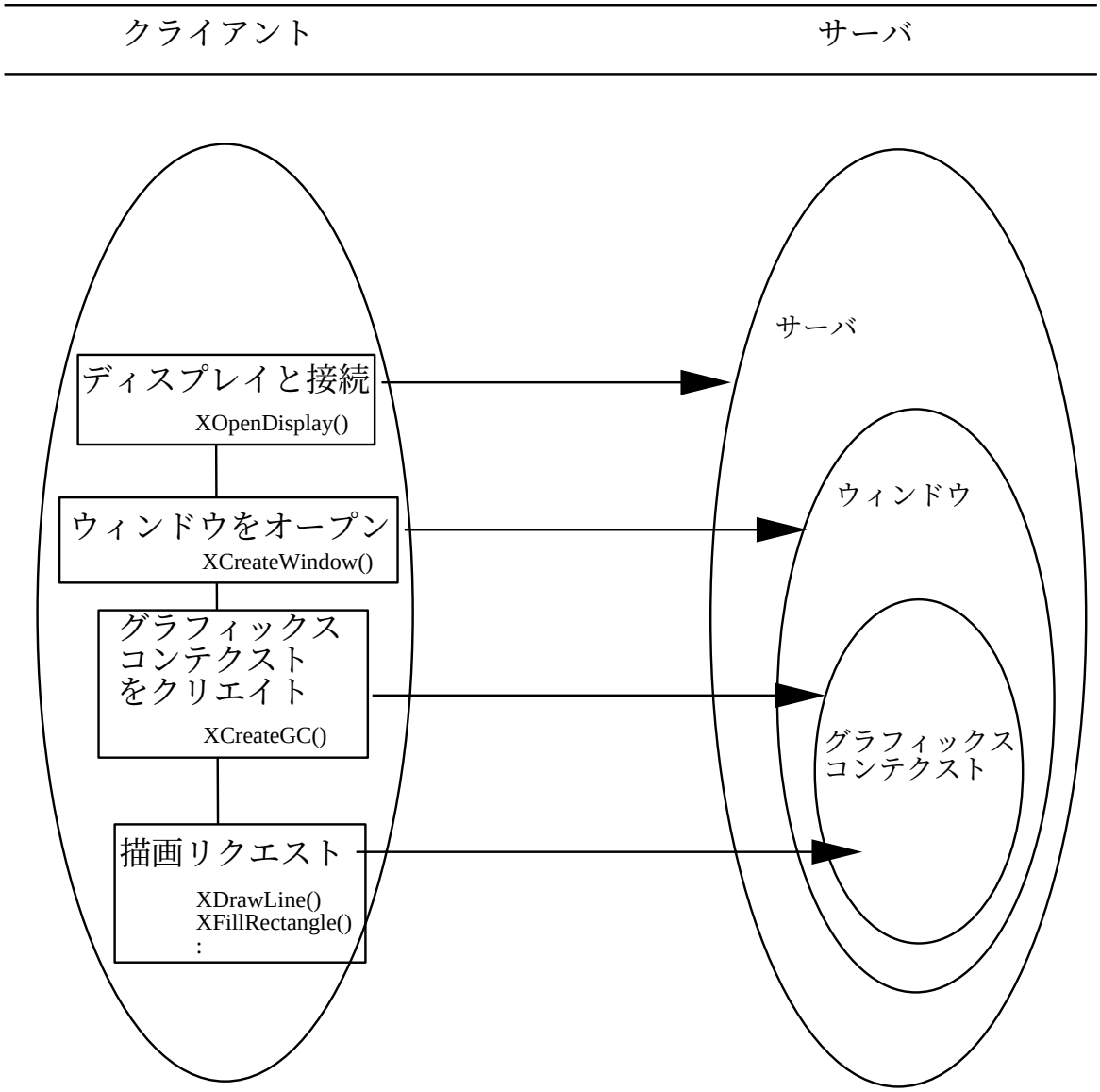
OpenGL 拡張の X Window System

- IRIX 5.X の X サーバ (Xsgi) は、OpenGL 拡張されている
- IRIX 5.X の aux ライブラリは、Xlib を使って実装している
- OpenGL の Graphical User Interface (GUI) は X Toolkit と Motif で作成できる

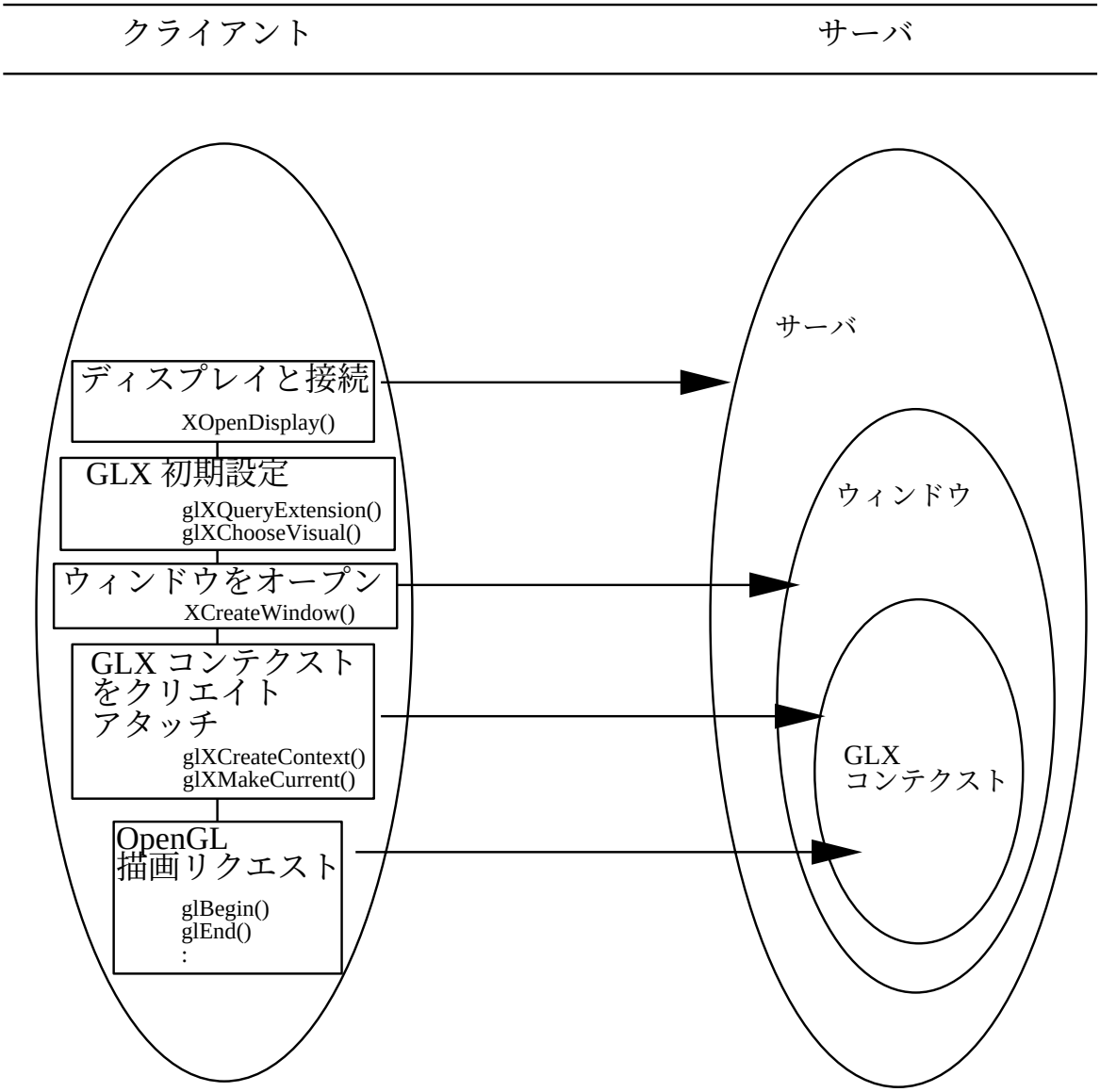


X Window System の描画の仕組み

- 描画に必要な属性をサーバ側にグラフィックス・コンテクストとして用意、そのグラフィックス・コンテクストを使って描画する
- OpenGL は GLX コンテクスト



X Window System で OpenGL の描画を行なう仕組み



A.2　X Window 用の OpenGL 拡張コマンド (GLX)

GLX 初期設定

- Bool glXQueryExtension(Display *dpy, int *errorBase, int *eventBase)
  - GLX 拡張がサポートされているかを確認、サポートされていれば真が返る
  - dpy は、X サーバとのコネクション
  - errorBase は、GLX 拡張コマンドのエラー・コードが入る領域
  - eventBase は、GLX 拡張コマンドのイベント・コードが入る領域
- Bool glXQueryVersion(Display *dpy, int *major, int *minor)
  - GLX 拡張のバージョン番号を返す
  - dpy は、X サーバとのコネクション
  - major は、GLX 拡張のメジャー・バージョン番号が入る領域
  - minor は、GLX 拡張のマイナー・バージョン番号が入る領域
- XVisualInfo *glXChooseVisual(Display *dpy, int screen, int *attribList)
  - 指定されたアトリビュートリストにマッチするビジュアルを返す
  - dpy は、X サーバとのコネクション
  - screen は、スクリーン番号
  - attribList は、アトリビュートリスト

GLX 初期設定

- int glXGetConfig(Display *dpy, XVisualInfo *vis, int attrib, int *value)
  - X Window での GL レンダリングのコンフィグ情報を得る
  - dpy は、X サーバとのコネクション
  - vis は、確認するビジュアル
  - attrib は、求めたいビジュアル属性
  - value は、求めた値が入る領域
- attrib の一覧

GLX_USE_GL	OpenGL をサポートしていれば真
GLX_BUFFER_SIZE	カラー・バッファのビット数
GLX_LEVEL	フレーム・バッファのレベル 正: より手前 0: 標準 負: より奥
GLX_RGBA	カラー・バッファが RGBA なら真
GLX_DOUBLEBUFFER	カラー・バッファが前後の組なら真
GLX_STEREO	カラー・バッファが左右の組なら真
GLX_AUX_BUFFERS	利用可能な補助カラー・バッファの数
GLX_RED_SIZE	赤のビット数
GLX_GREEN_SIZE	緑のビット数
GLX_BLUE_SIZE	青のビット数
GLX_ALPHA_SIZE	アルファのビット数
GLX_DEPTH_SIZE	デプス・バッファのビット数
GLX_STENCIL_SIZE	ステンシル・バッファのビット数
GLX_ACCUM_RED_SIZE	アキュムレーション・バッファの赤のビット数
GLX_ACCUM_GREEN_SIZE	アキュムレーション・バッファの緑のビット数
GLX_ACCUM_BLUE_SIZE	アキュムレーション・バッファの青のビット数
GLX_ACCUM_ALPHA_SIZE	アキュムレーション・バッファのアルファのビット数

GLX レンダリングコンテクストの管理

- GLXContext glXCreateContext(Display *dpy, XVisualInfo *vis, GLXContext shareList, Bool direct)
  - 新しく GLX レンダリングコンテクストをクリエイト
  - dpy は、X サーバとのコネクション
  - vis は、確認するビジュアル
  - shareList は、ディスプレイ・リストを共有するコンテクスト
  - direct は、ダイレクト・レンダリングのとき真
- Bool glXMakeCurrent(Display *dpy, GLXDrawable drawable, GLXContext ctx)
  - ウインドウあるいは GLX ピックスマップに GLX コンテクストをアタッチ
  - dpy は、X サーバとのコネクション
  - drawable は、GLX ドローワブル (X Window ID 或は GLX ピックスマップ)
  - ctx は、GLX コンテクスト
- void glXDestroyContext(Display *dpy, GLXContext ctx)
  - GLX コンテクストをデストロイ
  - dpy は、X サーバとのコネクション
  - ctx は、デストロイする GLX コンテクスト

GLX レンダリングコンテクストの問い合わせとコピー

- Bool glXIsDirect(Display *dpy, GLXContext ctx)
  - ダイレクトレンダリングが可能になっているかを確認
  - dpy は、X サーバとのコネクション
  - ctx は、GLX コンテクスト
- GLXContext glXGetCurrentContext(void)
  - カレント・コンテクストを返す
- GLXDrawable glXGetCurrentDrawable(void)
  - カレント・ドローワブルを返す
- void glXCopyContext(Display *dpy, GLXContext src, GLXContext dst, GLuint mask)
  - レンダリングコンテクストをコピー
  - dpy は、X サーバとのコネクション
  - src は、コピー元のコンテクスト
  - dst は、コピー先のコンテクスト
  - mask は、コピーするときのマスク

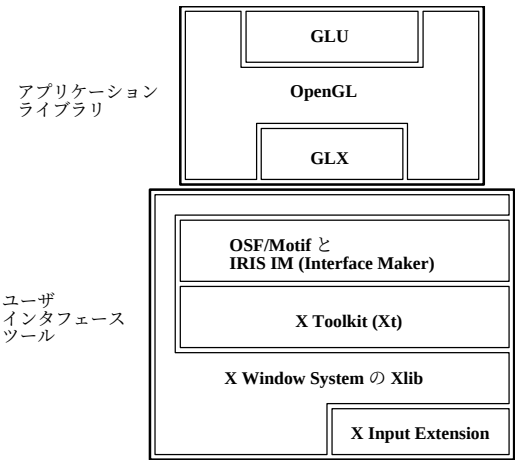
GL と X との同期

- void glXWaitGL(void)
  – GL の実行が終了するのを待つ
- void glXWaitX(void)
  – X の実行が終了するのを待つ

A.3 X Toolkit と Motif の概要

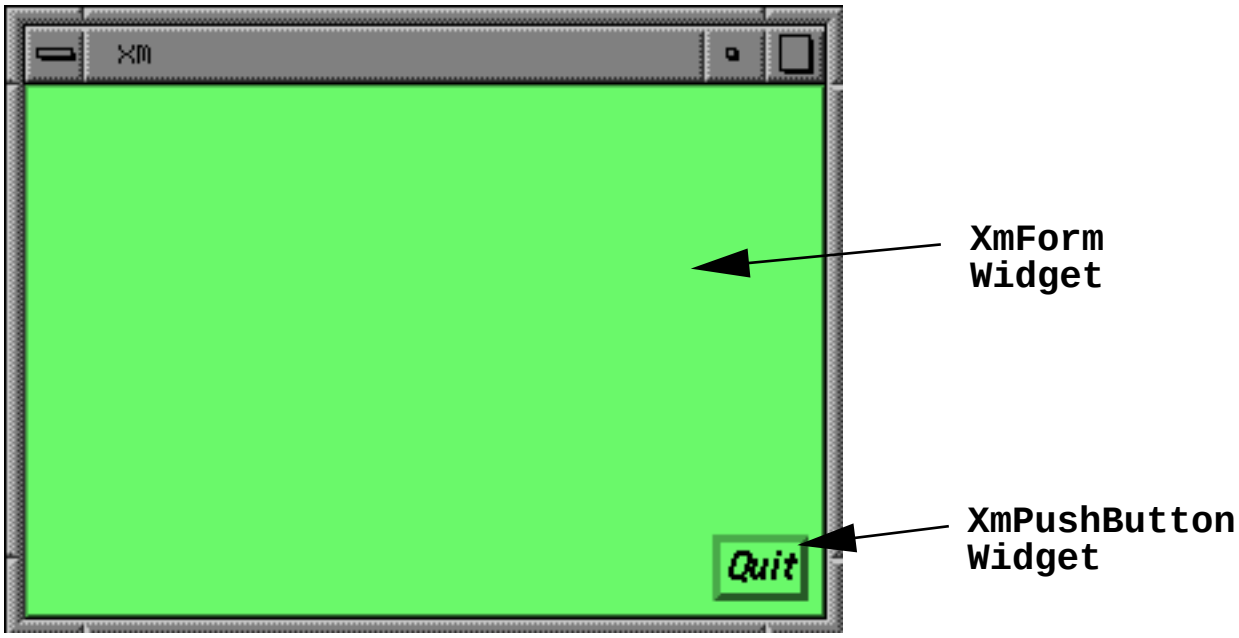
Motif とは

- OpenGL の Graphical User Interface (GUI) は X Toolkit と Motif で作成できる

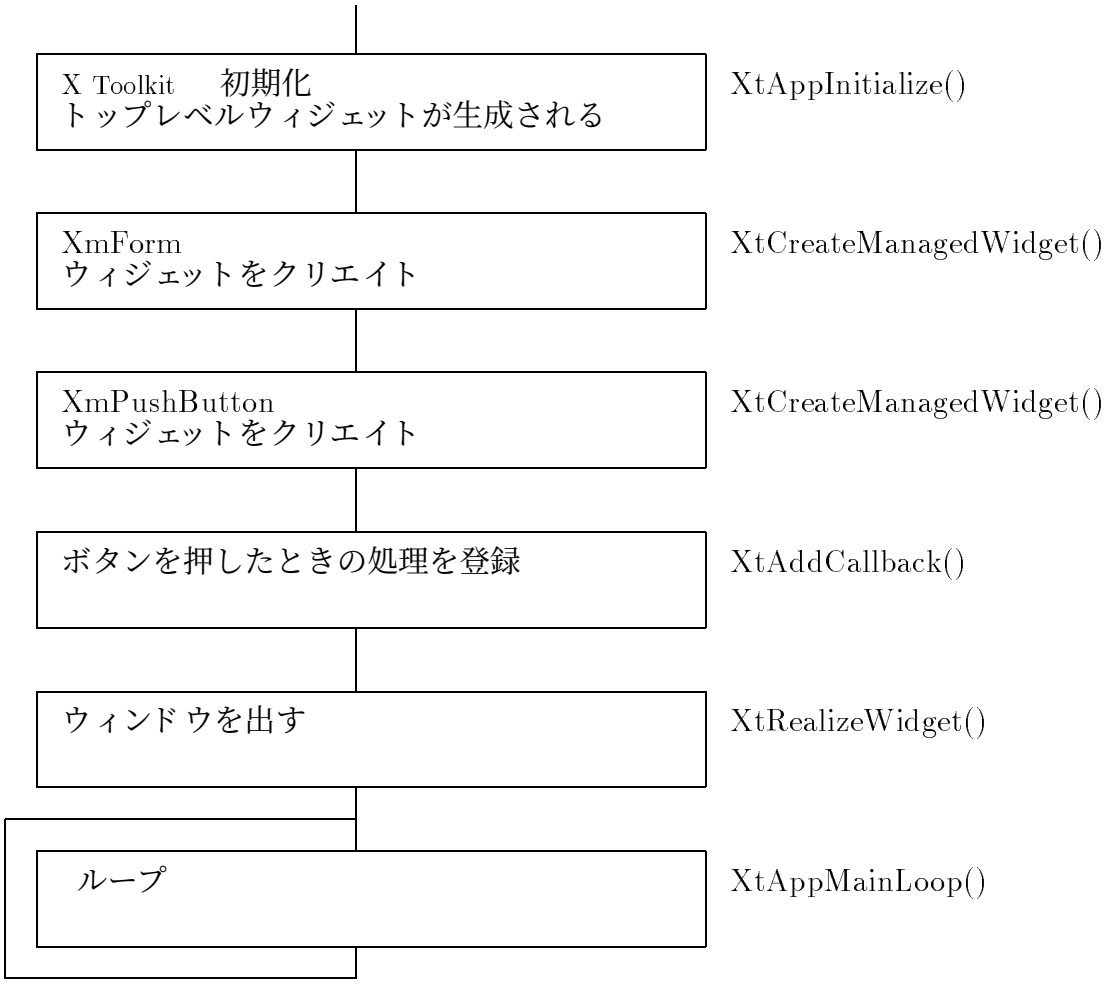


- GUI (Graphical User Interface) の部品をウィジェットと呼ぶ
- X Toolkit (Xt) は、ウィジェットを扱う仕組みを提供
- Motif は、ウィジェット・セット

toplevel – XmForm – XmPushButton



Motif プログラミングの流れ



Motif プログラミングの手順

- 国際化/地域対応化
  XtSetLanguageProc(NULL, NULL, NULL)
  環境変数 LANG でリソースをコントロールできるようになる。
- ボタンのラベルなどをリソースとして設定

  (1) X Toolkit 初期化のときに設定する方法
  ```
  static String fallbackResources[] = { … }
  XtAppInitialize(…, fallbackResources, …)
  ```

  (2) ウィジェットをクリエイトする時に設定する方法
  ```
  XtSetArg(args, …)
  XtCreateManagedWidget(…, args, …)
  ```

  (3) リソースファイルで設定する方法
  /usr/lib/X11/app-defaults ディレクトリ
  日本語用リソースファイルは
  /usr/lib/X11/ja_JP.EUC/app-defaults ディレクトリ

  (4) 個別に設定する方法
  ```
  XtSetArg( args, ... )
  XtSetValues( widget, args, ... )
  ```

- ボタンを押したときの処理をコールバック・ルーチンとして登録する。
  XtAddCallback()

```
 1 /*
 2  * xm.c
 3  *
 4  * Motif のサンプルプログラム
 5  *
 6  *     toplevel - XmForm - XmPushButton
 7  *
 8  * cc -o xm xm.c -lXm -lXt -lX11
 9  */
10 #include <X11/Intrinsic.h>
11 #include <X11/Xm/Xm.h>
12 #include <X11/Xm/Form.h>
13 #include <X11/Xm/PushB.h>
14
15 void quit(Widget w, XtPointer data, XtPointer callData)
16 {
17     exit(0);
18 }
19
20 int
21 main(int argc, char *argv[])
22 {
23     Widget toplevel;
24     Widget form, pushButton, label;
25     Widget message;
26     XtAppContext app;
27     Arg args[16];
28     Cardinal n;
29     static String fallbackResources[] = {
30  "*geometry: 300x200+100+100",
31  "*foreground: black",
32  "*background: PaleGreen",
33  "*fontList: -adobe-helvetica-*--14-*",
34  NULL
35     };
36
37    /*
38     * 国際化/地域対応化
39     */
40     XtSetLanguageProc(NULL, NULL, NULL);
41
42    /*
43     * X Toolkit を初期化
44     */
```

```
45     toplevel = XtAppInitialize(&app, "XmSample", NULL, 0, &argc, argv,
46  fallbackResources, NULL, 0);
47
48    /*
49     * XmForm ウィジェットをクリエイト
50     */
51     form = XtCreateManagedWidget("form", xmFormWidgetClass,
52  toplevel, NULL, 0);
53    /*
54     * XmPushButton ウィジェットをクリエイト
55     *     XmForm ウィジェットの右下隅にアタッチ
56     */
57     n = 0;
58     XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
59     XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
60     XtSetArg(args[n], XmNrightOffset, 5); n++;
61     XtSetArg(args[n], XmNbottomOffset, 5); n++;
62     XtSetArg(args[n], XmNlabelString,
63  XmStringCreate("Quit", XmSTRING_DEFAULT_CHARSET)); n++;
64     pushButton = XtCreateManagedWidget("pushButton", xmPushButtonWidgetClass,
65  form, args, n);
66     XtAddCallback(pushButton, XmNactivateCallback, quit, message);
67
68     XtRealizeWidget(toplevel);
69
70     XtAppMainLoop(app);
71     return 0;
72 }
73
```

<演習>

1. Motif のサンプルプログラムを実行して下さい。

```
% vi xm.c
% make xm
% xm
```

A.4 Motif と OpenGL の混合プログラミング

描画用ウィジェットに OpenGL で描画を行なう

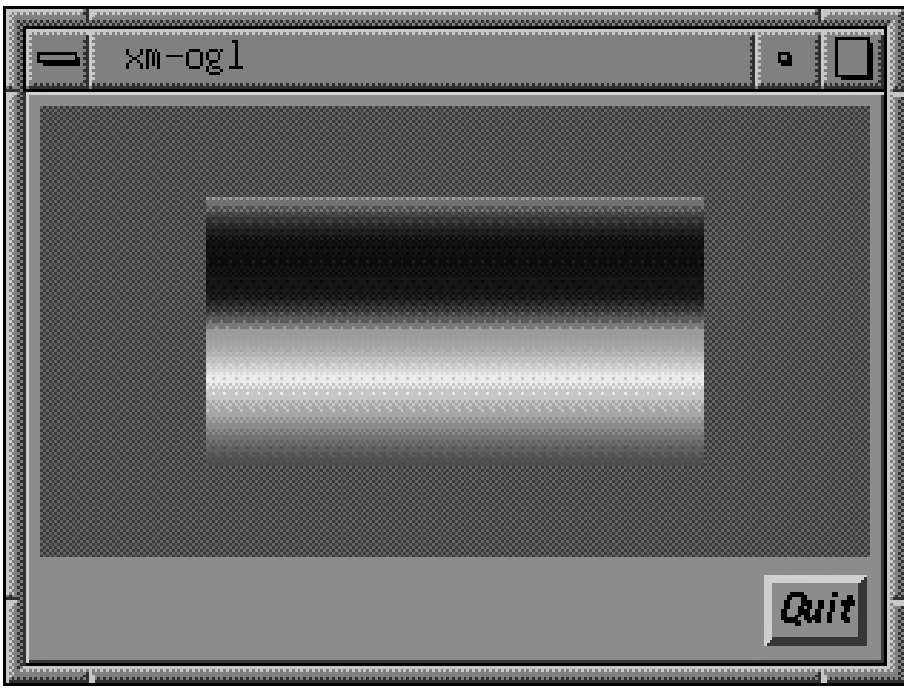
- IRIX 5.x が提供する描画用ウィジェット
  - XmDrawingArea ウィジェット
    Motif 描画用ウィジェット：通常 Xlib の 2D 描画用のウィジェット
  - GLwMDrawingArea ウィジェット
    Motif OpenGL 描画用ウィジェット：Motif ウィジェット・セットと組み合わせて使用
  - GLwDrawingArea ウィジェット
    汎用 OpenGL 描画用ウィジェット： Motif 以外のウィジェット・セット (例えば Athena ウィジェット・セット) とも組み合わせて使用可

`Motif` ウィジェット・セットのときは、上記すべてと組み合わせ可
`Athena` ウィジェット・セットのときは、`GLwDrawingArea` ウィジェットを使用

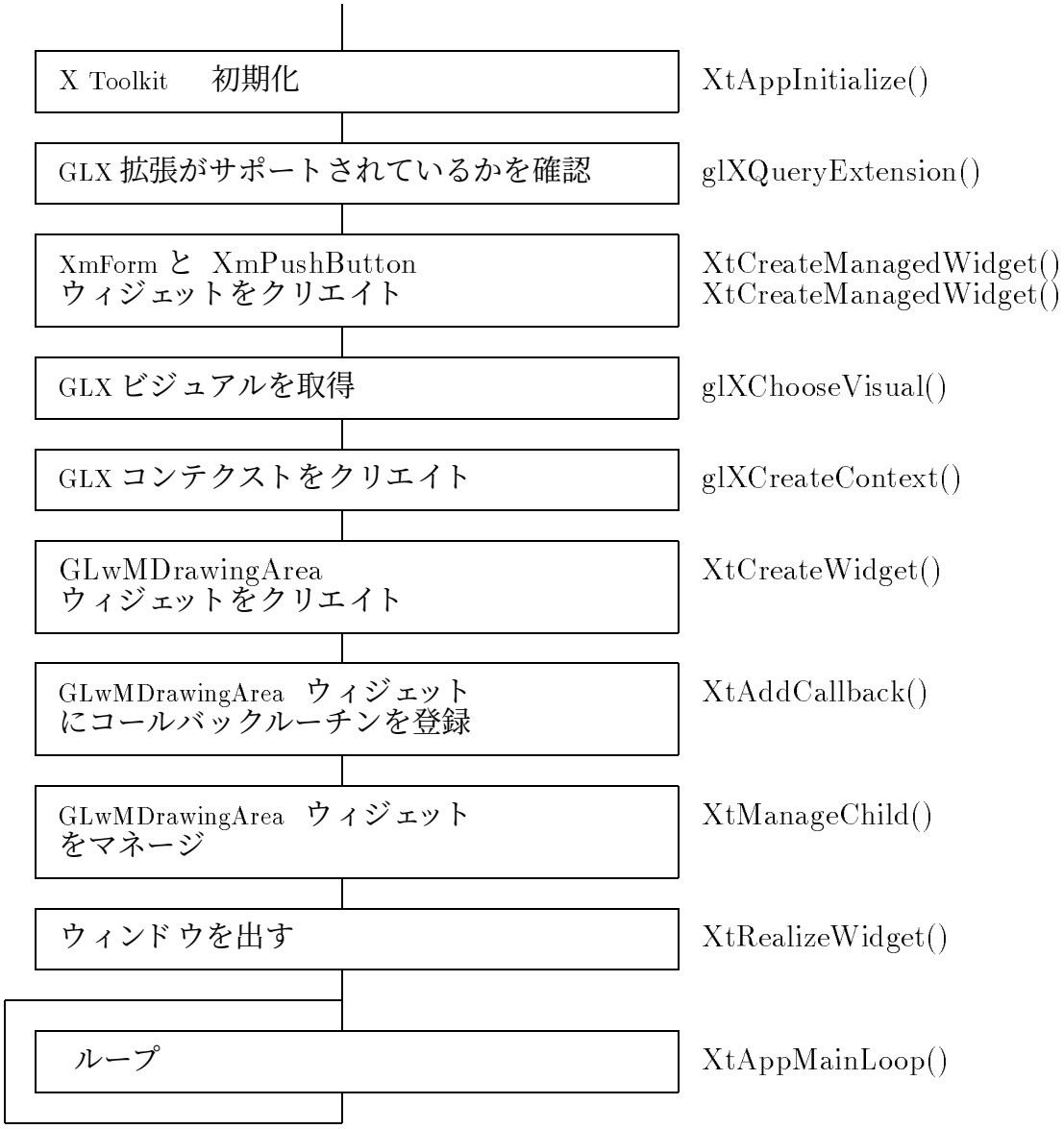
Motif と OpenGL の混合プログラミング実行例

- Motif のウィジェットと OpenGL 用のウィジェットの混合プログラミングの例

```
                    GLwMDrawingArea
                  /
toplevel -  XmForm
                  \
                    XmPushButton
```



Motif と OpenGL の混合プログラミングの流れ

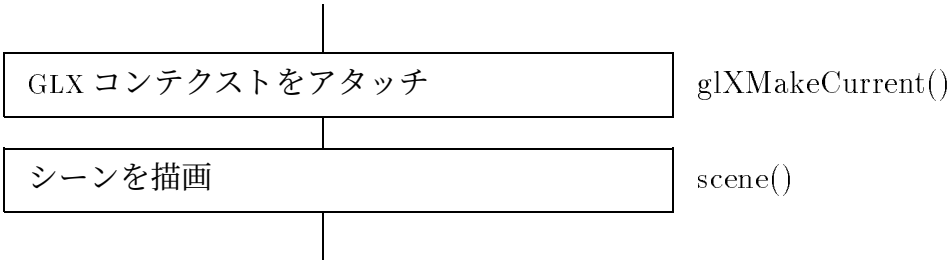


Motif と OpenGL の混合プログラミングの手順

- GLwMDrawingArea ウィジェットのコールバック

`GLwNexposeCallback`	再描画	`drawScene()`
`GLwNresizeCallback`	リサイズ	`resize()`
`GLwNinputCallback`	入力	`input()`
`GLwNginitCallback`	初期化	`ginit()`

- 初期化ルーチンで OpenGL のマトリックスを初期化する
- 再描画ルーチンで GLX コンテクストをアタッチしてからシーンを描画する



- リサイズルーチンでリサイズ後のウィンドウサイズにビューポートを更新する
- GLwMDrawingArea ウィジェットで入力イベントが発生すると入力ルーチンが起動される

```
 1 /*
 2  * xm-ogl.c
 3  *
 4  * Motif と OpenGL 混合のサンプルプログラム
 5  *
 6  *
 7  *                          GLwMDrawingArea
 8  *                        /
 9  *     toplevel - XmForm
10  *                        \
11  *                          XmPushButton
12  *
13  * cc -o xm-ogl xm-ogl.c -lGLw -lGL -lXm -lXt -lX11
14  */
15 #include <X11/Intrinsic.h>
16 #include <X11/Xm/Xm.h>
17 #include <X11/Xm/Form.h>
18 #include <X11/Xm/PushB.h>
19 #include <GL/GLwMDrawA.h>
20 #include <GL/glx.h>
21
22 Display *dpy;
23 GLXContext context;
24
25 /*
26  * シーンを描画
27  */
28 void scene()
29 {
30    GLfloat x, y;
31    int i;
32    static GLfloat raingow[][3] = {
33  {0.933, 0.509, 0.933}, /* violet */
34  {0.294, 0.000, 0.509}, /* indigo */
35  {0.000, 0.000, 1.000}, /* blue   */
36  {0.000, 1.000, 0.000}, /* green  */
37  {1.000, 1.000, 0.000}, /* yellow */
38  {1.000, 0.647, 0.000}, /* orange */
39  {1.000, 0.000, 0.000}, /* red    */
40     };
41
42     x = 0.6;
43     y = 0.6;
44     for (i = 0; i < 6; i++) {
```

```
45  glBegin(GL_POLYGON);
46    glColor3fv(raingow[i]);
47    glVertex2f(-x, y);
48    glVertex2f( x, y);
49    glColor3fv(raingow[i+1]);
50    glVertex2f( x, y-0.2);
51    glVertex2f(-x, y-0.2);
52  glEnd();
53  y = y - 0.2;
54     }
55 }
56
57 void ginit(Widget w, XtPointer data, XtPointer callData)
58 {
59     /*
60      * GLX コンテキストをアタッチ
61      */
62     glXMakeCurrent(dpy, XtWindow(w), context);
63
64     /*
65      * マトリックスを初期化
66      */
67     glMatrixMode(GL_PROJECTION);
68     glLoadIdentity();
69     glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
70     glMatrixMode(GL_MODELVIEW);
71     glLoadIdentity();
72 }
73
74 void drawScene(Widget w, XtPointer data, XtPointer callData)
75 {
76     /*
77      * GLX コンテキストをアタッチ
78      */
79     glXMakeCurrent(dpy, XtWindow(w), context);
80
81     glPushMatrix();
82
83     glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
84     glClear(GL_COLOR_BUFFER_BIT);
85
86     /*
87      * シーンを描画
88      */
```

```
 89     scene();
 90
 91     glPopMatrix();
 92     glFlush();
 93 }
 94
 95 void resize(Widget w, XtPointer data, XtPointer callData)
 96 {
 97     Dimension width, height;
 98
 99     /*
100      * GLX コンテキストをアタッチ
101      */
102     glXMakeCurrent(dpy, XtWindow(w), context);
103
104     if (XtIsRealized(w)) {
105
106  /*
107   * Widget がリアライズしていれば、幅と高さを取得
108   */
109         XtVaGetValues(w, XtNwidth, &width, XtNheight, &height, NULL);
110
111  /*
112   * ビューポートを更新
113   */
114         glViewport(0, 0, (GLint)width, (GLint)height);
115     }
116 }
117
118 void quit(Widget w, XtPointer data, XtPointer callData)
119 {
120     exit(0);
121 }
122
123 int
124 main(int argc, char *argv[])
125 {
126     Widget toplevel;
127     Widget form, pushButton, glx;
128     XtAppContext app;
129     XVisualInfo *visualInfo;
130     Arg args[16];
131     Cardinal n;
132     Dimension offset;
```

```
133     static int attributes[] = { GLX_RGBA,
134  GLX_RED_SIZE,   1,
135  GLX_GREEN_SIZE, 1,
136  GLX_BLUE_SIZE,  1,
137  None };
138     static String fallbackResources[] = {
139  "*geometry: 300x200+100+100",
140  "*foreground: black",
141  "*background: PaleGreen",
142  "*fontList: -adobe-helvetica-*--14-*",
143  NULL
144     };
145
146     XtSetLanguageProc(NULL, NULL, NULL);
147
148     toplevel = XtAppInitialize(&app, "XmSample", NULL, 0, &argc, argv,
149  fallbackResources, NULL, 0);
150     dpy = XtDisplay(toplevel);
151
152    /*
153     * GLX 拡張がサポートされているかを確認
154     */
155     if (glXQueryExtension(dpy, NULL, NULL) == NULL) exit(1);
156
157     form = XtCreateManagedWidget("form", xmFormWidgetClass,
158  toplevel, NULL, 0);
159     n = 0;
160     XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
161     XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
162     XtSetArg(args[n], XmNrightOffset, 5); n++;
163     XtSetArg(args[n], XmNbottomOffset, 5); n++;
164     XtSetArg(args[n], XmNlabelString,
165  XmStringCreate("Quit", XmSTRING_DEFAULT_CHARSET)); n++;
166     pushButton = XtCreateManagedWidget("pushButton", xmPushButtonWidgetClass,
167  form, args, n);
168     XtAddCallback(pushButton, XmNactivateCallback, quit, NULL);
169
170     /*
171      * OpenGL 描画 Widget を pushButton の上に配置するために
172      * pushButton の高さを取得
173      */
174     n = 0;
175     XtSetArg(args[n], XtNheight, &offset); n++;
176     XtGetValues(pushButton, args, n);
```

```
177     offset += 10;
178
179     /*
180      * GLX ビジュアルを取得
181      */
182     visualInfo = glXChooseVisual(dpy, DefaultScreen(dpy), attributes);
183
184     /*
185      * GLX コンテキストをクリエイト
186      */
187     context = glXCreateContext(dpy, visualInfo, None, GL_TRUE);
188
189     /*
190      * OpenGL 描画 Widget を pushButton の上に配置
191      * XmForm ウィジェットの上下左右にアタッチ
192      */
193     n = 0;
194     XtSetArg(args[n], GLwNvisualInfo, visualInfo); n++;
195     XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
196     XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
197     XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
198     XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
199     XtSetArg(args[n], XmNleftOffset, 5); n++;
200     XtSetArg(args[n], XmNrightOffset, 5); n++;
201     XtSetArg(args[n], XmNtopOffset, 5); n++;
202     XtSetArg(args[n], XmNbottomOffset, offset); n++;
203     XtSetArg(args[n], XmNtraversalOn, True); n++;
204     glx = XtCreateWidget("glx", glwMDrawingAreaWidgetClass,
205 form, args, n);
206
207     /*
208      * 再描画コールバックを登録
209      */
210     XtAddCallback(glx, GLwNexposeCallback, drawScene, NULL);
211
212     /*
213      * リサイズコールバックを登録
214      */
215     XtAddCallback(glx, GLwNresizeCallback, resize, NULL);
216
217     /*
218      * 初期化コールバックを登録
219      */
220     XtAddCallback(glx, GLwNginitCallback, ginit, NULL);
```

```
221
222     /*
223      * OpenGL 描画 Widget をマネージ
224      */
225     XtManageChild(glx);
226
227     XtRealizeWidget(toplevel);
228
229     XtAppMainLoop(app);
230     return 0;
231 }
232
```

<演習>

1. Motif と OpenGL 混合のサンプルプログラムを実行して下さい。

```
% vi xm-ogl.c
% make xm-ogl
% xm-ogl
```

付 録B　X/Motif との混合プログラミング (2) 応用編

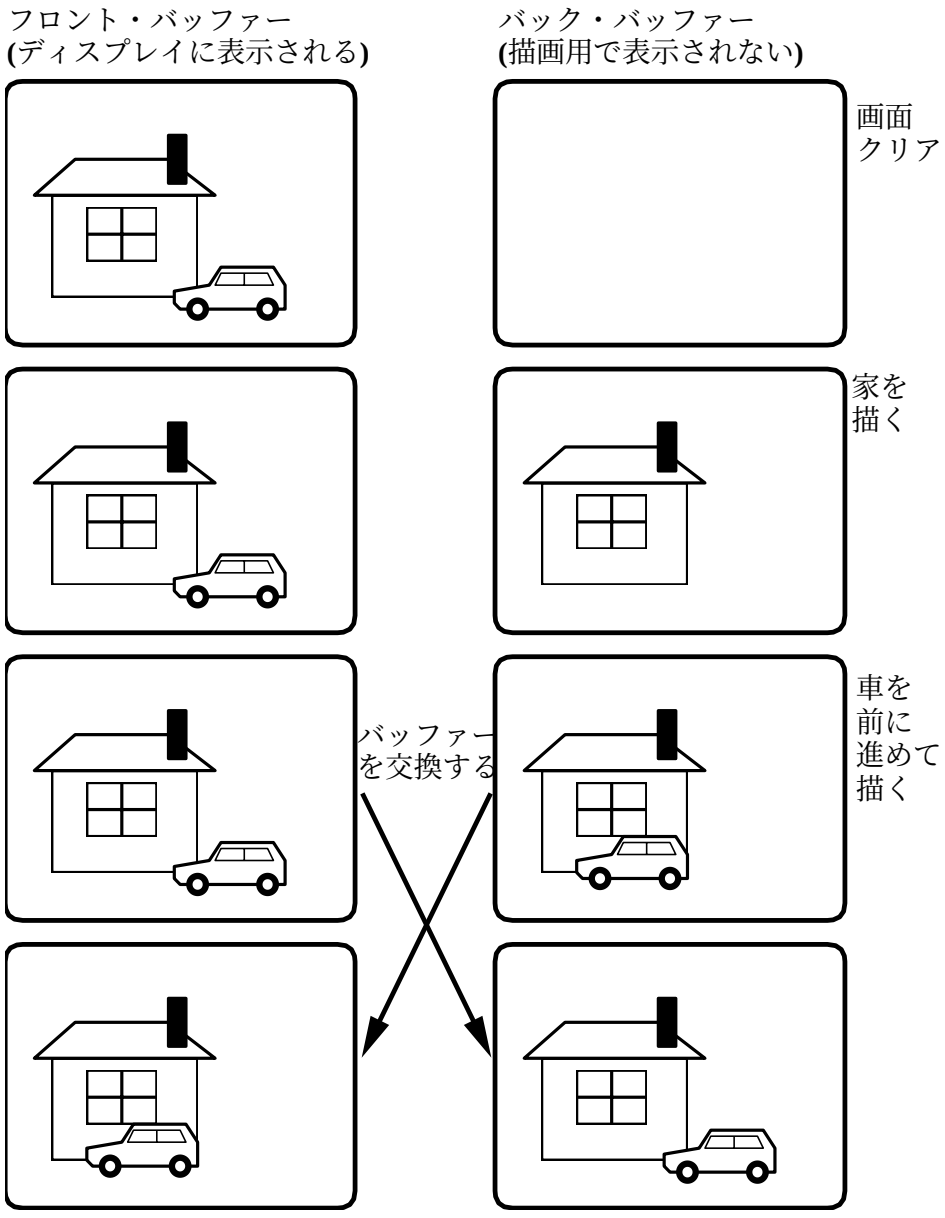
本章の目的

- アニメーションプログラミング
- 入力イベントの取り扱い
- pixmap を使ったオフスクリーンレンダリング
- X フォントの利用方法

B.1　アニメーションプログラミング

ダブルバッファ

- void glXSwapBuffers(Display *dpy, GLXDrawable drawable)
  - フロントバッファとバックバッファを切り換える
  - dpy は、X サーバとのコネクション
  - drawable は、バッファを取り替える X Window ID



アニメーションプログラミングの流れ

- WorkProc を使う
- WorkProc は、真を return するまで繰り返し自分を起動する仕組み
- XtAppAddWorkProc( ) で WorkProc を登録する

```
void drawScene(Widget w, XtPointer data, XtPointer callData)
{
    :
    《再描画》
    :
    glXSwapBuffers(XtDisplay(w), XtWindow(w));
}

Boolean workProc(XtPointer data)
{
    :
    《シーンを再計算》
    :
    drawScene((Widget)data, NULL, NULL);
    return False;
}

main(int argc, char *argv[])
{
    static int attributes[] = { GLX_DOUBLEBUFFER, GLX_RGBA, None };

    visualInfg = glXChooseVisual(dpy, DefaultScreen(dpy), attributes);
    XtSetArg(args[n], GLwNvisualInfo, visualInfo); n++;
    :
    glx = XtCreateManagedWidget("glx", glwMDrawingAreaWidgetClass,
                    parent, args, n);
    :
    XtAppAddWorkProc(app, workProc, glx);
    XtAppMainLoop(app);
}
```

アニメーションプログラミングの流れ

- GLX ビジュアルを取得するとき、GLX_DOUBLEBUFFER を指定
- ダブルバッファ・モードで GLwMDrawingArea ウィジェットをクリエイト
- 再描画の後、glXSwapBuffers() でバックバッファとフロントバッファを取り替える
  そして、偽を return する

```
 1 /*
 2  * rotate.c
 3  *
 4  * workProc を使ったアニメーション
 5  *
 6  *                          GLwMDrawingArea
 7  *                         /
 8  *     toplevel - XmForm
 9  *                         \
10  *                          XmPushButton
11  *
12  * cc -o rotate rotate.c -lGLw -lGL -lXm -lXt -lX11
13  */
14 #include <X11/Intrinsic.h>
15 #include <X11/Xm/Xm.h>
16 #include <X11/Xm/Form.h>
17 #include <X11/Xm/PushB.h>
18 #include <GL/GLwMDrawA.h>
19 #include <GL/glx.h>
20
21 GLfloat angle = 0.0;
22
23 Display *dpy;
24 GLXContext context;
25
26 void scene()
27 {
28    GLfloat x, y;
29    int i;
30    static GLfloat raingow[][3] = {
31  {0.933, 0.509, 0.933}, /* violet */
32  {0.294, 0.000, 0.509}, /* indigo */
33  {0.000, 0.000, 1.000}, /* blue   */
34  {0.000, 1.000, 0.000}, /* green  */
35  {1.000, 1.000, 0.000}, /* yellow */
36  {1.000, 0.647, 0.000}, /* orange */
37  {1.000, 0.000, 0.000}, /* red    */
38     };
39
40     x = 0.6;
41     y = 0.6;
42     for (i = 0; i < 6; i++) {
43  glBegin(GL_POLYGON);
44    glColor3fv(raingow[i]);
```

```
45    glVertex2f(-x, y);
46    glVertex2f( x, y);
47    glColor3fv(raingow[i+1]);
48    glVertex2f( x, y-0.2);
49    glVertex2f(-x, y-0.2);
50  glEnd();
51  y = y - 0.2;
52     }
53 }
54
55 void ginit(Widget w, XtPointer data, XtPointer callData)
56 {
57     glXMakeCurrent(dpy, XtWindow(w), context);
58
59     glMatrixMode(GL_PROJECTION);
60     glLoadIdentity();
61     glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
62     glMatrixMode(GL_MODELVIEW);
63     glLoadIdentity();
64 }
65
66 void drawScene(Widget w, XtPointer data, XtPointer callData)
67 {
68     glXMakeCurrent(dpy, XtWindow(w), context);
69
70     glPushMatrix();
71
72     glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
73     glClear(GL_COLOR_BUFFER_BIT);
74
75     glRotatef(angle, 0.0, 0.0, 1.0);
76
77     scene();
78
79     glPopMatrix();
80
81     /*
82      * スワップバッファ
83      */
84     glXSwapBuffers(dpy, XtWindow(w));
85 }
86
87 /*
88  * WorkProc
```

```
 89  */
 90 Boolean workProc(XtPointer data)
 91 {
 92     angle += 0.5; if (angle >= 360.0) angle = 0.0;
 93     drawScene((Widget)data, NULL, NULL);
 94     return False;
 95 }
 96
 97 void resize(Widget w, XtPointer data, XtPointer callData)
 98 {
 99     Dimension width, height;
100
101     /*
102      * GLX コンテキストをアタッチ
103      */
104     glXMakeCurrent(dpy, XtWindow(w), context);
105
106     if (XtIsRealized(w)) {
107
108  /*
109   * Widget がリアライズしていれば、幅と高さを取得
110   */
111         XtVaGetValues(w, XtNwidth, &width, XtNheight, &height, NULL);
112
113  /*
114   * ビューポートを更新
115   */
116         glViewport(0, 0, (GLint)width, (GLint)height);
117     }
118 }
119
120 void quit(Widget w, XtPointer data, XtPointer callData)
121 {
122     exit(0);
123 }
124
125 int
126 main(int argc, char *argv[])
127 {
128     Widget toplevel;
129     Widget form, pushButton, glx;
130     XtAppContext app;
131     XVisualInfo *visualInfo;
132     Arg args[16];
```

```
133     Cardinal n;
134     Dimension offset;
135  /* ダブルバッファ */
136     static int attributes[] = { GLX_DOUBLEBUFFER,
137  GLX_RGBA,
138  GLX_RED_SIZE,   1,
139  GLX_GREEN_SIZE, 1,
140  GLX_BLUE_SIZE,  1,
141  None };
142     static String fallbackResources[] = {
143  "*geometry: 300x200+100+100",
144  "*foreground: black",
145  "*background: PaleGreen",
146  "*fontList: -adobe-helvetica-*--14-*",
147  NULL
148     };
149
150     XtSetLanguageProc(NULL, NULL, NULL);
151
152     toplevel = XtAppInitialize(&app, "XmSample", NULL, 0, &argc, argv,
153  fallbackResources, NULL, 0);
154     dpy = XtDisplay(toplevel);
155
156    /*
157     * GLX 拡張がサポートされているかを確認
158     */
159     if (glXQueryExtension(dpy, NULL, NULL) == NULL) exit(1);
160
161     form = XtCreateManagedWidget("form", xmFormWidgetClass,
162  toplevel, NULL, 0);
163     n = 0;
164     XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
165     XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
166     XtSetArg(args[n], XmNrightOffset, 5); n++;
167     XtSetArg(args[n], XmNbottomOffset, 5); n++;
168     XtSetArg(args[n], XmNlabelString,
169  XmStringCreate("Quit", XmSTRING_DEFAULT_CHARSET)); n++;
170     pushButton = XtCreateManagedWidget("pushButton", xmPushButtonWidgetClass,
171  form, args, n);
172     XtAddCallback(pushButton, XmNactivateCallback, quit, NULL);
173
174     n = 0;
175     XtSetArg(args[n], XtNheight, &offset); n++;
176     XtGetValues(pushButton, args, n);
```

```
177     offset += 10;
178
179     visualInfo = glXChooseVisual(dpy, DefaultScreen(dpy), attributes);
180     context = glXCreateContext(dpy, visualInfo, None, GL_TRUE);
181
182     n = 0;
183     XtSetArg(args[n], GLwNvisualInfo, visualInfo); n++;
184     XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
185     XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
186     XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
187     XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
188     XtSetArg(args[n], XmNleftOffset, 5); n++;
189     XtSetArg(args[n], XmNrightOffset, 5); n++;
190     XtSetArg(args[n], XmNtopOffset, 5); n++;
191     XtSetArg(args[n], XmNbottomOffset, offset); n++;
192     XtSetArg(args[n], XmNtraversalOn, True); n++;
193     glx = XtCreateWidget("glx", glwMDrawingAreaWidgetClass,
194  form, args, n);
195
196     XtAddCallback(glx, GLwNexposeCallback, drawScene, NULL);
197     XtAddCallback(glx, GLwNresizeCallback, resize, NULL);
198     XtAddCallback(glx, GLwNginitCallback, ginit, NULL);
199
200     XtManageChild(glx);
201
202     XtRealizeWidget(toplevel);
203
204     /*
205      * WorkProc を登録
206      */
207     XtAppAddWorkProc(app, workProc, glx);
208
209     XtAppMainLoop(app);
210     return 0;
211 }
212
```

<演習>

1. workProc を使ったアニメーションのサンプルプログラムを実行して下さい。

```
% vi rotate.c
% make rotate
% rotate
```

B.2 入力イベントの取り扱い

入力イベントプログラミングの流れ

- X Window System のイベント・ハンドラを使う
- GLwNinputCallback に input() が登録されているので、何か入力があると input() が起動される
- 第三引数 callData から、イベントを取得して解析

```
void input(Widget w, XtPointer data, XtPointer callData)
{
    XmDrawingAreaCallbackStruct *cs;
    char buf[4];
    KeySym keysym;

    cs = (XmDrawingAreaCallbackStruct *)callData;
    switch (cs->event->type)
    {
    case KeyPress:
        :
        《キーボードが押されたときの処理》
        :
        break;

    case KeyRelease:
        :
        《キーボードが離されたときの処理》
        :
        break;

    case ButtonPress:
        :
        《マウスのボタンが押されたときの処理》
        :
        break;

    case ButtonRelease:
```

```
        :
        《マウスのボタンが離されたときの処理》
        :
        break;

    case MotionNotify:
        :
        《マウス・ドラッグのときの処理》
        :
        break;
    }
}
```

どのキーが押されたかを判定: `XLookupString()`

マウスのボタンの番号: `cs->event->xbutton.button`

マウス・ドラッグ時の X ピクセル値: `cs->event->xbutton.x`
マウス・ドラッグ時の Y ピクセル値: `cs->event->xbutton.y`

```
 1 /*
 2  * event.c
 3  *
 4  * 入力イベント
 5  *   マウス左ドラッグで移動
 6  *   マウス中クリックでホームポジション
 7  *   Esc で終了
 8  *
 9  *
10  *                          GLwMDrawingArea
11  *                        /
12  *     toplevel - XmForm
13  *                        \
14  *                          XmPushButton
15  *
16  * cc -o event event.c -lGLw -lGL -lXm -lXt -lX11
17  */
18 #include <X11/Intrinsic.h>
19 #include <X11/keysym.h>
20 #include <X11/Xm/Xm.h>
21 #include <X11/Xm/Form.h>
22 #include <X11/Xm/PushB.h>
23 #include <GL/GLwMDrawA.h>
24 #include <GL/glx.h>
25
26 #define LEFT_BUTTON   1
27 #define MIDDLE_BUTTON 2
28
29 GLfloat angle = 0.0;
30 GLfloat orgx = 0.0;
31 GLfloat orgy = 0.0;
32
33 Display *dpy;
34 GLXContext context;
35
36 /*
37  * 入力イベント
38  */
39 void input(Widget w, XtPointer data, XtPointer callData)
40 {
41     XmDrawingAreaCallbackStruct *cs;
42     char buf[4];
43     KeySym keysym;
44     static GLfloat x, y, oldx, oldy;
```

```
45     static Boolean leftButton = False;
46
47     cs = (XmDrawingAreaCallbackStruct *)callData;
48     switch (cs->event->type)
49     {
50     case KeyPress:
51     case KeyRelease:
52  if (XLookupString((XKeyEvent *)cs->event, buf, 1, &keysym, NULL) == 1)
53      if (keysym == XK_Escape) exit(0);
54  break;
55
56     case ButtonPress:
57  x = cs->event->xbutton.x;
58  y = cs->event->xbutton.y;
59  if (cs->event->xbutton.button == LEFT_BUTTON) leftButton = True;
60  if (cs->event->xbutton.button == MIDDLE_BUTTON) {
61      orgx = 0.0;
62      orgy = 0.0;
63  }
64  break;
65
66     case ButtonRelease:
67  if (cs->event->xbutton.button == LEFT_BUTTON) leftButton = False;
68  break;
69
70     case MotionNotify:
71  if (leftButton) {
72      oldx = x;
73      oldy = y;
74      x = cs->event->xbutton.x;
75      y = cs->event->xbutton.y;
76      orgx = orgx + (x - oldx)*0.005;
77      orgy = orgy - (y - oldy)*0.005;
78  }
79     }
80 }
81
82 void scene()
83 {
84    GLfloat x, y;
85    int i;
86    static GLfloat raingow[][3] = {
87  {0.933, 0.509, 0.933}, /* violet */
88  {0.294, 0.000, 0.509}, /* indigo */
```

```
 89  {0.000, 0.000, 1.000}, /* blue   */
 90  {0.000, 1.000, 0.000}, /* green  */
 91  {1.000, 1.000, 0.000}, /* yellow */
 92  {1.000, 0.647, 0.000}, /* orange */
 93  {1.000, 0.000, 0.000}, /* red    */
 94     };
 95
 96     x = 0.6;
 97     y = 0.6;
 98     for (i = 0; i < 6; i++) {
 99  glBegin(GL_POLYGON);
100    glColor3fv(raingow[i]);
101    glVertex2f(-x, y);
102    glVertex2f( x, y);
103    glColor3fv(raingow[i+1]);
104    glVertex2f( x, y-0.2);
105    glVertex2f(-x, y-0.2);
106  glEnd();
107  y = y - 0.2;
108     }
109 }
110
111 void ginit(Widget w, XtPointer data, XtPointer callData)
112 {
113     glXMakeCurrent(dpy, XtWindow(w), context);
114
115     glMatrixMode(GL_PROJECTION);
116     glLoadIdentity();
117     glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
118     glMatrixMode(GL_MODELVIEW);
119     glLoadIdentity();
120 }
121
122 void drawScene(Widget w, XtPointer data, XtPointer callData)
123 {
124     glXMakeCurrent(dpy, XtWindow(w), context);
125
126     glPushMatrix();
127
128     glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
129     glClear(GL_COLOR_BUFFER_BIT);
130
131     glTranslatef(orgx, orgy, 0.0);
132     glRotatef(angle, 0.0, 0.0, 1.0);
```

```
133
134     scene();
135
136     glPopMatrix();
137     glXSwapBuffers(dpy, XtWindow(w));
138 }
139
140 /*
141  * WorkProc
142  */
143 Boolean workProc(XtPointer data)
144 {
145     angle += 0.5; if (angle >= 360.0) angle = 0.0;
146     drawScene((Widget)data, NULL, NULL);
147     return False;
148 }
149
150 void resize(Widget w, XtPointer data, XtPointer callData)
151 {
152     Dimension width, height;
153
154     /*
155      * GLX コンテキストをアタッチ
156      */
157     glXMakeCurrent(dpy, XtWindow(w), context);
158
159     if (XtIsRealized(w)) {
160
161  /*
162   * Widget がリアライズしていれば、幅と高さを取得
163   */
164         XtVaGetValues(w, XtNwidth, &width, XtNheight, &height, NULL);
165
166  /*
167   * ビューポートを更新
168   */
169         glViewport(0, 0, (GLint)width, (GLint)height);
170     }
171 }
172
173 void quit(Widget w, XtPointer data, XtPointer callData)
174 {
175     exit(0);
176 }
```

```
177
178 int
179 main(int argc, char *argv[])
180 {
181     Widget toplevel;
182     Widget form, pushButton, glx;
183     XtAppContext app;
184     XVisualInfo *visualInfo;
185     Arg args[16];
186     Cardinal n;
187     Dimension offset;
188     static int attributes[] = { GLX_DOUBLEBUFFER,
189  GLX_RGBA,
190  GLX_RED_SIZE,   1,
191  GLX_GREEN_SIZE, 1,
192  GLX_BLUE_SIZE,  1,
193  None };
194     static String fallbackResources[] = {
195  "*geometry: 300x200+100+100",
196  "*foreground: black",
197  "*background: PaleGreen",
198  "*fontList: -adobe-helvetica-*--14-*",
199  NULL
200     };
201
202     XtSetLanguageProc(NULL, NULL, NULL);
203
204     toplevel = XtAppInitialize(&app, "XmSample", NULL, 0, &argc, argv,
205  fallbackResources, NULL, 0);
206     dpy = XtDisplay(toplevel);
207
208    /*
209     * GLX 拡張がサポートされているかを確認
210     */
211     if (glXQueryExtension(dpy, NULL, NULL) == NULL) exit(1);
212
213     form = XtCreateManagedWidget("form", xmFormWidgetClass,
214  toplevel, NULL, 0);
215     n = 0;
216     XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
217     XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
218     XtSetArg(args[n], XmNrightOffset, 5); n++;
219     XtSetArg(args[n], XmNbottomOffset, 5); n++;
220     XtSetArg(args[n], XmNlabelString,
```

```
221  XmStringCreate("Quit", XmSTRING_DEFAULT_CHARSET)); n++;
222     pushButton = XtCreateManagedWidget("pushButton", xmPushButtonWidgetClass,
223  form, args, n);
224     XtAddCallback(pushButton, XmNactivateCallback, quit, NULL);
225
226     n = 0;
227     XtSetArg(args[n], XtNheight, &offset); n++;
228     XtGetValues(pushButton, args, n);
229     offset += 10;
230
231     visualInfo = glXChooseVisual(dpy, DefaultScreen(dpy), attributes);
232     context = glXCreateContext(dpy, visualInfo, None, GL_TRUE);
233
234     n = 0;
235     XtSetArg(args[n], GLwNvisualInfo, visualInfo); n++;
236     XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM); n++;
237     XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM); n++;
238     XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
239     XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
240     XtSetArg(args[n], XmNleftOffset, 5); n++;
241     XtSetArg(args[n], XmNrightOffset, 5); n++;
242     XtSetArg(args[n], XmNtopOffset, 5); n++;
243     XtSetArg(args[n], XmNbottomOffset, offset); n++;
244     XtSetArg(args[n], XmNtraversalOn, True); n++;
245     glx = XtCreateWidget("glx", glwMDrawingAreaWidgetClass,
246  form, args, n);
247
248     /*
249      * 入力コールバックを登録
250      */
251     XtAddCallback(glx, GLwNinputCallback, input, NULL);
252
253     XtAddCallback(glx, GLwNexposeCallback, drawScene, NULL);
254     XtAddCallback(glx, GLwNresizeCallback, resize, NULL);
255     XtAddCallback(glx, GLwNginitCallback, ginit, NULL);
256
257     XtManageChild(glx);
258
259     XtRealizeWidget(toplevel);
260
261     /*
262      * WorkProc を登録
263      */
264     XtAppAddWorkProc(app, workProc, glx);
```

```
265
266     XtAppMainLoop(app);
267     return 0;
268 }
269
```

<演習>

1. 入力イベントのサンプルプログラムを実行して下さい。

```
% vi event.c
% make event
% event
```

B.3 X 機能の利用

オフスクリーンレンダリング

- pixmap を使ってメモリー上でレンダリングを実行する
  - pixmap は X サーバーのビジュアルに制限されるので、任意の解像度はできない
- GLXPixmap glXCreateGLXPixmap(Display *dpy, XVisualInfo *vis, Pixmap pixmap)
  - オフスクリーン GLX レンダリングエリアをクリエイト
  - dpy は、X サーバとのコネクション
  - vis は、レンダリング領域の構造を定義したビジュアル
  - pixmap は、オフスクリーンのカラー・バッファとして使用する X ピックスマップ
- void glXDestroyGLXPixmap(Display *dpy, GLXPixmap pix)
  - GLX ピックスマップをデストロイ
  - dpy は、X サーバとのコネクション
  - pix は、デストロイする GLX ピックスマップ

X font の利用

- void glXUseXFont(Font font, int first, int count, int listBase)
  - X のフォントからビットマップディスプレイリストをクリエイト
  - font は、キャラクタ・グリフ (X Window System のフォント)
  - first は、最初のグリフの指標
  - count は、グリフの数
  - listBase は、最初に生成するディスプレイ・リストの指標

サブルーチンのまとめ

GLX コマンド情報の取得

```
Bool glXQueryExtension( Display *dpy, int *errorBase, int *eventBase )
Bool glXQueryVersion( Display *dpy, int *major, int *minor )
int glXGetConfig( Display *dpy, XVisualInfo *vis, int attrib, int *value )
XVisualInfo* glXChooseVisual( Display *dpy, int screen, int *attribList )
```

GLX コンテクスト

```
GLXContext glXCreateContext( Display *dpy, XVisualInfo *vis,
                             GLXContext shareList, Bool direct )
Bool glXMakeCurrent( Display *dpy, GLXDrawable drawable, GLXContext ctx )
void glXDestroyContext( Display *dpy, GLXContext ctx )
Bool glXIsDirect( Display *dpy, GLXContext ctx )
GLXContext glXGetCurrentContext( void )
GLXDrawable glXGetCurrentDrawable( void )
void glXCopyContext( Display *dpy, GLXContext src,
                     GLXContext dst, GLuint mask )
```

ピクスマップ

```
GLXPixmap glXCreateGLXPixmap( Display *dpy, XVisualInfo *vis,
                              Pixmap pixmap )
void glXDestroyGLXPixmap( Display *dpy, GLXPixmap pix )
```

同期

```
void glXWaitGL( void )
void glXWaitX( void )
```

バッファ

```
void glXSwapBuffers( Display *dpy, GLXDrawable drawable )
```

フォント

```
void glXUseXFont( Font font, int first, int count, int listBase )
```

付 録C　Windows NTによる OpenGLプログラミング

本章の目的

- OpenGL のための Windows NT の初期設定方法を理解する
- MFC を利用した場合の OpenGL プログラミングを理解する
- カラー・パレット管理を理解する
- アニメーション・プログラムの作り方を理解する

C.1 Windows NT OpenGLプログラミングの基礎

必要なハードウェア・リソース

- CPU は浮動小数点の速いものを選ぶ
  - Pentium 互換 CPU の中には浮動小数点の遅いものがあるので注意
  - MMX は重視しなくて良い
- メイン・メモリーは Visual C++/VisualStudio など開発環境が快適に動作することを第一基準にする
- ビデオ・カードは OpenGL 対応ドライバのある 3D 機能搭載のものを選ぶ
  - ビデオメモリーは 8MB を基準に考える
  - パッケージ・アプリケーション開発の場合は 2MB 以下のモデルでカラー・パレット管理が正しく行われるかどうかのチェックが重要
- 開発ツールの大規模化を考慮し 4GB 以上の SCSI ディスクが望ましい
  - Visual C++プログラミングはデバッグ情報などでディスクを数 MB 単位で多量に消費する

3次元グラフィックス・アクセラレーション・ボード

- ラスター処理を中心に高速化
  現行の商品の多くは座標変換などのジオメトリ処理は CPU が行なう
  – ハイエンドではジオメトリ処理サポートもある
  – トライアングル・セットアップはローエンドでもサポート
- OpenGL ドライバーの有無を確認して選ぶ
  – OpenGL ドライバーには MCD と ICD がある
  – MCD は Microsoft が OpenGL ドライバを簡単に開発できるようにした方式
  – MCD は Microsoft の提供する独自拡張機能を利用できる
  – ICD はハードベンダーが OpenGL すべての処理をコントロールできる
  – ICD はチューニングに有利だが、拡張機能の実装がハードベンダーに依存する

必要なソフトウェア・リソース

- Windows NT 3.5 以上のバージョンに OpenGL 実行環境が付属する
- Windows 95 OSR2(OEM Service Release 2) 以降のバージョンに OpenGL 実行環境が付属する
- OSR2 以前の Windows 95 の場合は Microsoft から OpenGL 環境を ftp で入手する

  `ftp://ftp.microsoft.com/softlib/mslfiles/Opengl95.exe`

- Visual C++ があれば OpenGL プログラミングは必要十分である
  - GUI のインタラクティブな設計が可能
  - Visual C++はライブラリ、例題、オンライン・マニュアルを完備
  - Borland C++など他のコンパイラでも OpenGL プログラミングは可能だが例題やドキュメントの整備に問題があり不利である
  - Borland C++用の GLUT ライブラリは配布されている
- Windows OpenGL プログラミングの例題や解説文書は次のものが詳しい
  - Microsoft の Developer Network の CD-ROM(MSDN 会員)
  - Windpws NT の SDK や DDK(MSDN Professional 会員以上)
  - Microsoft のホームページにも WWW(http://www.microsoft.com)
  - Microsoft Journal の記事 (ときどき)
- 参考になる書籍
  - OpenGL Superbible,
    Wright, Macmillan Computer, ISBN #1571690735
    * 包括的で辞書的な解説書
  - OpenGL Programming for Windows 95 & Windows NT,
    Fosner, Addison-Wesley, ISBN #0201407094
    * 簡潔で読みやすい入門書
  - 3D Graphics File Formats,
    RULE, Addison-Wesley, ISBN #0201488353
    * 3 次元グラフィックスのファイルフォーマットと OpenGL OCX プログラミング

OpenGLプログラミング・スタイル

- GLUT、あるいはAUXを使った開発
  - UNIX、OS/2、Macなどへの移植性では最も優れる
  - ミニ・プログラムやプロトタイピング向きで、
    実用的プログラムを作るのには機能が不足
- MFC(Microsoft Foundation Class)を使った開発
  - OpenGLを直接サポートするクラスはVisual C++には含まれていない
  - ActiveX/OCXプログラミングも簡単にできる
- MFCを使わないC言語ベースの開発
  - Visual C++製品でMicrosoftの提供する例題はほとんどこのスタイル
  - Visual C++の統合環境が有効に使えず、コーディング部分が多くなる
  - ここでは解説しない

GLUTを使ったプログラミング

- 本書の例題はほとんど無変更で使うことができる
- GLUT は UAX に含まれるイメージの読み込み機能はない
  - 例えば SGI イメージなら fstimage を使う
    http://www.webcity.co.jp/ matumot/fstimage.zip
- ライブラリは以下のものを指定する
  - opengl32.lib, glu32.lib, glut.lib
- インクルードファイルは gl.h と glu.h は指定せず glut.h のみを指定する
  - windows.h, gl.h, glu.h は glut.h の中で適切な順番で呼ばれる
  - 移植性を高めることができる
    "#ifndef WIN32"を使わなくていい
- AppWizard ではコンソールアプリケーションを指定してプロジェクトを生成する

C.2 MFCプログラミング

MFCプログラムの基本構造

- MFC プログラミングでは 4 つのクラスの実装が基本になる
  - アプリケーションクラス (CWinApp) : 以下の 3 つのクラスの統括
  - フレームウィンドウクラス (CFrameWnd) : タイトルバー、境界線などを管理
  - ビュークラス (CView) : ウインドウ内部のグラフィックスなどを管理
  - ドキュメントクラス (CDocument) : ファイル入出力など含むデータ管理
- OpenGL 関連の描画やイベント処理はビュークラスが中心となる
  - ビュークラスのメンバー関数から OpenGL 関数がコールされる
  - カラーパレット管理やアニメーション処理には複数のクラスのインタラクションが必要になる

MFC と OpenGL

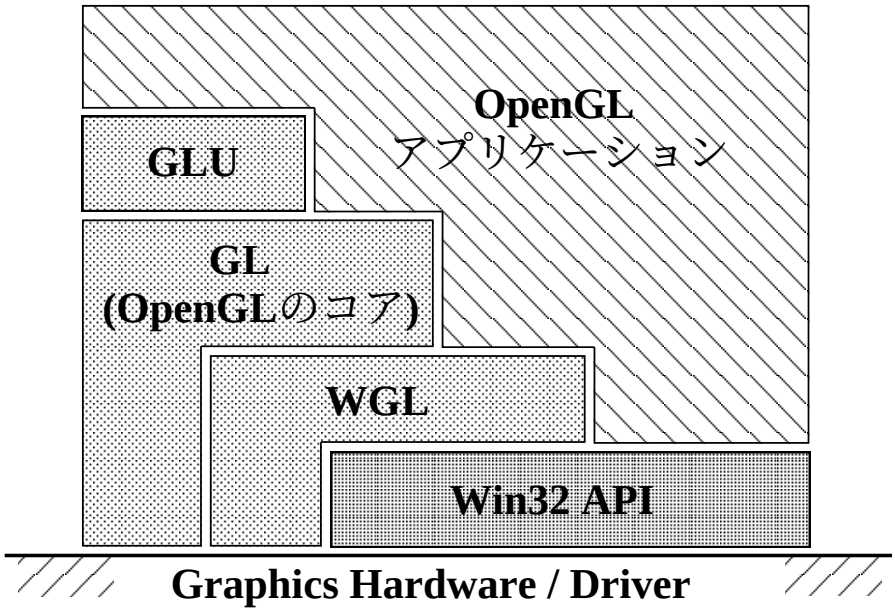
- Visual C++には OpenGL をカプセル化したようなクラスは含まれない
- MFC を使った例題は Microsoft が MSDN の CDROM で提供している
  - Dennis Crain の MYGL プログラム (”Windows NT OpenGL:Getting started”)
  - Dele Rogerson の EazyGL プログラム (”OpenGL III: Building an OpenGL C++ Class”)
- OpenGL の処理をクラス化しなくてもよい
  - ビュークラスの中から OpenGL を使用している C の関数群を呼べばよい

MFCによるOpenGLプログラミングのポイント

- AppWizard でプロジェクトを生成
  (以下の例では Simple というプロジェクトを使った)
- ヘッダーとライブラリの指定
- 次の順で枠組みを作る。
  1. ヘッダーとライブラリの指定
  2. ウインドウ生成前のモード設定
  3. ピクセル・フォーマットの設定
  4. カラーパレット管理
  5. ビュークラスの各種メンバー関数でのイベント処理
- アニメーションプログラム
  – ビュークラスとフレームウィンドウクラスのオブジェクト間でメッセージ・パッシングを行う

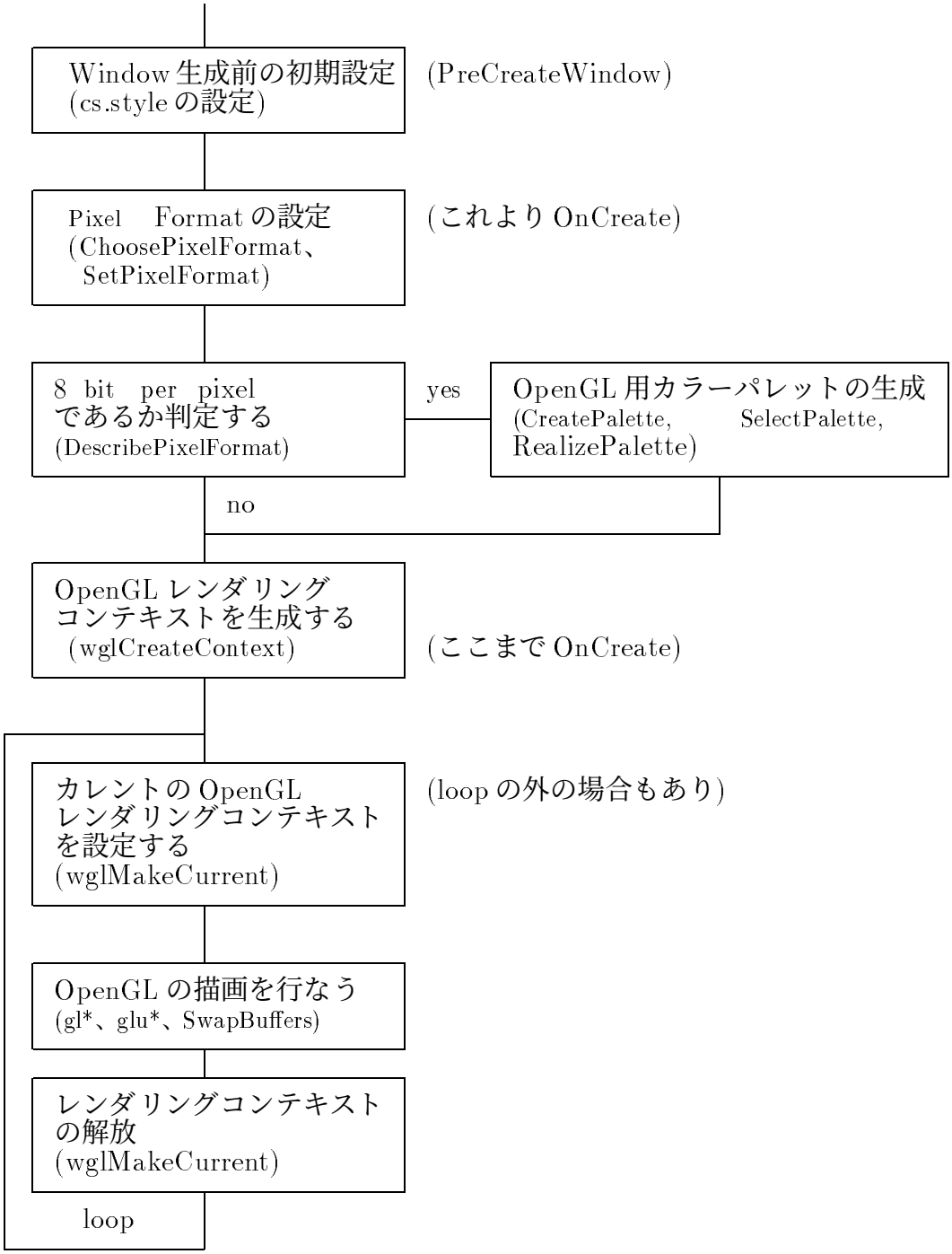
ヘッダーとライブラリの追加

- Visual C++でプロジェクトを作成した後に行う
  - Visual C++が stdfx.h というヘッダーファイルを自動生成する
- stdfx.h に OpenGL 用のヘッダーファイルを追加
  - #include <GL/gl.h>
  - #include <GL/glu.h>
- リンクのオプションでライブラリを追加する
  - opengl32.lib、glu32.lib
  - glaux.lib や glut.lib は必要に応じて加える
- Windows の OpenGL ライブラリ構成は次の通り



プログラミングの流れ

- OpenGL 描画に必要な設定をまとめると次の流れ図のようになる
- 以下この図に従い OpenGL のための設定を行う



C.3　OpenGL のための初期設定

ウインドウ生成前の初期設定 (PreCreateWindow 関数)

- OpenGL は、OpenGL のために設定したウインドウのクライアント・エリアにのみレンダリングできる
  – child window や sublings のクライアント・エリアにはレンダリングできない
- ビュークラスの PreCreateWindow メンバー関数で child window や siblings を clip する
  1. クラス・ウィザードのメッセージマップでビュークラスのクラス名を選択する
  2. オブジェクト ID のウィンドウからそのクラス名を選択する
  3. メッセージのウィンドウから PreCreateWindow を選び「関数の追加」ボタンをクリックする
  4. コード編集を選び以下のコードのようにする
     以下の例ではビュークラス名は CSimpleView とした

```
BOOL CSimpleView::PreCreateWindow(CREATESTRUCT& cs)
{
    cs.style = cs.style | WS_CLIPSIBLINGS | WS_CLIPCHILDREN;
    return CView::PreCreateWindow(cs);
}
```

- これが一番始めに必要な処理で、Windows に固有の設定
  – ActiveX/OCX プログラミングのときには不要

ピクセルフォーマットの設定（OnCreate関数）

- ピクセルフォーマットで描画ウインドウの属性を指定する
- ビュークラスの OnCreate 関数に実装する
- 次のものを適宜指定する
  - PFD_DRAW_TO_WINDOW
  - PFD_SUPPORT_OPENGL
  - PFD_TYPE_RGBA
  - PFD_TYPE_COLORINDEX
  - PFD_DOUBLEBUFFER
  - PFD_STEREO
  - PFD_DRAW_TO_BITMAP
  - PFD_SUPPORT_GDI
- ChoosePixelFormat 関数で最適なピクセルフォーマットを選択し、
  SetPixelFormat 関数でそのピクセルフォーマットに設定する
- ピクセルフォーマットは Windows NT 3.5 から GDI に導入された
- X でいうウィンドウのビジュアルの設定と同様
- 次のページにピクセルフォーマット設定関数の例を示す
  設定関数のヘッダーファイル (oglpre.h) は次の通り

```
extern BOOL SetupPixelFormat(HDC hdc);
extern BOOL CreateRGBPalette(HDC hdc, CPalette **ppCPalette);
```

```
#include "stdafx.h"
#include "oglPre.h"

BOOL SetupPixelFormat(HDC hDC)
{
    static PIXELFORMATDESCRIPTOR pfd = {
        sizeof(PIXELFORMATDESCRIPTOR),  // この pfd のサイズ
        1,                              // バージョン番号
        PFD_DRAW_TO_WINDOW |            // ウィンドウに描画する
          PFD_DOUBLEBUFFER |            // ダブルバッファリングを使う
          PFD_SUPPORT_OPENGL,           // OpenGL を使う
        PFD_TYPE_RGBA,                  // RGBA モードを使う
        24,                             // カラーバッファの深さは 24-bit
        0, 0, 0, 0, 0, 0,               // カラービットの指定（無視）
        0,                              // アルファプレーンはなし
        0,                              // アルファビットの指定（無視）
        0,                              // アキュムレーションバッファはなし
        0, 0, 0, 0,                     // アキュムレーションバッファbit 指定

        32,                             // z-buffer の深さ
        0,                              // ステンシルバッファーはなし
        0,                              // 補助のバッファーはなし
        PFD_MAIN_PLANE,                 // メインプレーンのみ
        0,                              // reserved
        0, 0, 0                         // layer masks ignored
    };
    int pixelformat;

    if ( (pixelformat = ChoosePixelFormat(hDC, &pfd)) == 0 )
    {
        TRACE("ChoosePixelFormat failed\n");
        return FALSE;
    }

    if (SetPixelFormat(hDC, pixelformat, &pfd) == FALSE)
    {
        TRACE("SetPixelFormat failed\n");
        return FALSE;
    }
    return TRUE;
}
```

C.4 カラー・パレットの管理

カラー・パレット管理の概要

- 8bpp(bit per pixel) の同時 256 色表示のディスプレイモードで RGBA モードを使用する場合に必要になる
  16bpp、24bpp のモードでは必要ない
- Visual C++ OpenGL の例題の GENGL プログラムが MFC を使わない場合のパレット管理法を示している
- MFC を使う場合のパレット管理については Dale Rogerson の “OpenGL II : Windows Palletes in RGB Mode” が詳しい
  - Windows では 1 つのシステム・パレットを各アプリケーションが共有する
  - 各アプリケーションは 1 つ以上の論理パレットを持つことができる
  - 論理パレットをシステムパレットにインストールするのはアプリケーションが行う
  - 論理パレットからシステム・パレットへのマッピングはシステムが管理する
    * システム・パレットの 0 から 9、246 から 255 の 20 色はシステム・カラーとして予約されており、変更できない
    * システム・カラーの存在のため、同じ色の論理パレットのインデックスと、システム・パレットのインデックスは一般的に異なる
    * 論理パレットのインデックスとシステム・パレットのインデックスが一致する場合、その論理パレットをアイデンティティー・パレットという
      アイデンティティー・パレットはインデックス変換計算がいらないため高速
    * OpenGL の 8bpp ディスプレイの RGB モード用の論理パレットはアイデンティティー・パレットの構成にすることができない
      8bpp の RGB モードは処理速度が遅くなる場合がある
- システム・パレットをモニタリングするツールがあるとデバッグに有効
  c.mos 氏作成の syspal プログラムが便利 (ftp://ftp.webcity.co.jp/OpenGL/)

カラーパレットの初期設定 (OnCreate関数)

- ピクセルフォーマットの指定後に必要があるか調べてパレットの初期化をする
  - DescribePixelFormat 関数で PIXELFORMATDISCRIPTOR 構造体の dwFlags フィールドを調べる
  - dwFlags フィールドを調べて PFD_NEED_PALETTE フラグがセットされていたらパレット管理が必要になる
- LOGPALETTE 構造体と色の配列のためのメモリーを確保する
- LOGPALETTE 構造体にパレットのバージョンとパレット・テーブルのエントリー数を指定する
- palPalEntry 配列に色を設定する
  - はじめに RGB 3-3-2 のフォーマットで色の配列を設定する
    RGB 3-3-2 とは Red、Green、Blue を各々3bit、3bit、2bit で表現するものである
    それぞれ 8 階調、8 階調、4 階調ということ
  - RGB 3-3-2 フォーマットの色の配列に、その中にシステム・カラーを 12 個、できるだけ近い色を置き換えて埋め込む
  - 論理パレットにシステム・カラー 20 色の埋め込んでおかないとシステム・パレットに確実にインストールされる色数は 236 色までである
    どの色が捨て去られるかわからない
- CreatePalette で論理パレットを作成する
- SelectPalette で論理パレットを設定する
- RealizePallete で論理パレットをシステムパレットにインストールする
- パレット設定関数の例を次のページから示すヘッダーファイルは前出の oglpre.h と同じである

```
#include "stdafx.h"
#include "oglPre.h"

static unsigned char threeto8[8] = {
    0, 0111>>1, 0222>>1, 0333>>1, 0444>>1, 0555>>1, 0666>>1, 0377
};
static unsigned char twoto8[4] = {
    0, 0x55, 0xaa, 0xff
};
static unsigned char oneto8[2] = {
    0, 255
};
static int defaultOverride[13] = {
    0, 3, 24, 27, 64, 67, 88, 173, 181, 236, 247, 164, 91
};

static PALETTEENTRY defaultPalEntry[20] = {
    { 0,   0,   0,    0 },
    { 0x80,0,   0,    0 },
    { 0,   0x80,0,    0 },
    { 0x80,0x80,0,    0 },
    { 0,   0,   0x80, 0 },
    { 0x80,0,   0x80, 0 },
    { 0,   0x80,0x80, 0 },
    { 0xC0,0xC0,0xC0, 0 },

    { 192, 220, 192,  0 },
    { 166, 202, 240,  0 },
    { 255, 251, 240,  0 },
    { 160, 160, 164,  0 },

    { 0x80,0x80,0x80, 0 },
    { 0xFF,0,   0,    0 },
    { 0,   0xFF,0,    0 },
    { 0xFF,0xFF,0,    0 },
    { 0,   0,   0xFF, 0 },
    { 0xFF,0,   0xFF, 0 },
    { 0,   0xFF,0xFF, 0 },
    { 0xFF,0xFF,0xFF, 0 }
};

static unsigned char
ComponentFromIndex(int i, UINT nbits, UINT shift)
{
    unsigned char val;

    val = (unsigned char) (i >> shift);
    switch (nbits) {

    case 1:
        val &= 0x1;
        return oneto8[val];

    case 2:
```

```
            val &= 0x3;
            return twoto8[val];

        case 3:
            val &= 0x7;
            return threeto8[val];

        default:
            return 0;
        }
    }

    BOOL
    CreateRGBPalette(HDC hDC, CPalette **ppCPalette)
    {
        PIXELFORMATDESCRIPTOR pfd;
        LOGPALETTE *pPal;
        int n, i;

        n = GetPixelFormat(hDC);
        DescribePixelFormat(hDC, n, sizeof(PIXELFORMATDESCRIPTOR), &pfd);

        if (!(pfd.dwFlags & PFD_NEED_PALETTE)) return FALSE;

        n = 1 << pfd.cColorBits;
        pPal = (PLOGPALETTE)LocalAlloc(LMEM_FIXED, sizeof(LOGPALETTE) +
                n * sizeof(PALETTEENTRY));
        pPal->palVersion = 0x300;
        pPal->palNumEntries = n;
        for (i=0; i<n; i++) {
            pPal->palPalEntry[i].peRed =
                    ComponentFromIndex(i, pfd.cRedBits, pfd.cRedShift);
            pPal->palPalEntry[i].peGreen =
                    ComponentFromIndex(i, pfd.cGreenBits, pfd.cGreenShift);
            pPal->palPalEntry[i].peBlue =
                    ComponentFromIndex(i, pfd.cBlueBits, pfd.cBlueShift);
            pPal->palPalEntry[i].peFlags = 0;
        }

        /* fix up the palette to include the default GDI palette */
        if ((pfd.cColorBits == 8)                           &&
            (pfd.cRedBits   == 3) && (pfd.cRedShift   == 0) &&
            (pfd.cGreenBits == 3) && (pfd.cGreenShift == 3) &&
            (pfd.cBlueBits  == 2) && (pfd.cBlueShift  == 6)
           ) {
            for (i = 1 ; i <= 12 ; i++)
                pPal->palPalEntry[defaultOverride[i]] = defaultPalEntry[i];
        }

        if (*ppCPalette) delete *ppCPalette;
        *ppCPalette = new CPalette;
        BOOL result = (*ppCPalette)->CreatePalette(pPal);

        LocalFree(pPal);
```

```
    return result;
}
```

- OnCreate 関数は次のようになる

```
int CSimpleView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CView::OnCreate(lpCreateStruct) == -1)
        return -1;

    m_pDC = GetDC();
    ASSERT(m_pDC->m_hDC);

    // ピクセルフォーマットの設定
    SetupPixelFormat(m_pDC->m_hDC);

    // カラーパレットの設定
    if (CreateRGBPalette(m_pDC->m_hDC, &m_pCPal))
    {
        CPalette* pOldPal = m_pDC->SelectPalette(m_pCPal, FALSE);
        UINT u = m_pDC->RealizePalette();
        if (u != 0) InvalidateRect(NULL, TRUE);
    }

    // OpenGL レンダリング・コンテキストの生成
    m_hrc = wglCreateContext(m_pDC->m_hDC);
    if (!m_hrc)
    {
        TRACE("wglCreateContext Failed %x\r\n", GetLastError());
        return -1;
    }

    if (!wglMakeCurrent(m_pDC->m_hDC, m_hrc))
    {
        TRACE("wglMakeCurrent Failed %d\r\n", GetLastError());;
        return -1;
    }

    return 0;
}
```

- ビュークラスの定義は次の通り

```
// Simplvw.h : interface of the CSimpleView class
//
/////////////////////////////////////////////////////////////////////////////

class CSimpleView : public CView
{
protected: // create from serialization only
    CSimpleView();
    DECLARE_DYNCREATE(CSimpleView)

// Attributes
```

```
public:
    CSimpleDoc* GetDocument();

    // for Animation
    void Tick(void);
    void Spin(BOOL);

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CSimpleView)
    public:
    virtual void OnDraw(CDC* pDC);  // overridden to draw this view
    virtual void OnInitialUpdate();
    protected:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CSimpleView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:
    CDC* m_pDC ;          // Device Context Class
    HGLRC m_hrc ;         // OpenGL Rendering Context
    CPalette* m_pCPal ;   // Palette
    BOOL m_spin ;         // アニメーション表示用
    float m_rotx, m_roty; // アニメーション表示用

// Generated message map functions
protected:
    //{{AFX_MSG(CSimpleView)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnSize(UINT nType, int cx, int cy);
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
    afx_msg void OnDestroy();
    afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
    afx_msg BOOL OnQueryNewPalette();
    afx_msg void OnSpinStart();
    afx_msg void OnSpinStop();
    afx_msg void OnUpdateSpinStop(CCmdUI* pCmdUI);
    afx_msg void OnUpdateSpinStart(CCmdUI* pCmdUI);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in Simplvw.cpp
inline CSimpleDoc* CSimpleView::GetDocument()
```

```
    { return (CSimpleDoc*)m_pDocument; }
#endif

//////////////////////////////////////////////////////////////////////////////
```

カラーパレット維持 (1)
(フレームウィンドウ・クラスの処理)

- 初期化時だけではなく他のアプリケーションにシステム・パレットを書き換えられたときなどにも処理する
- パレット関連のイベントはフレームウィンドウ・クラスのオブジェクトに送られる
  – WM_PALETTECHAGED と WM_QUERYNEWPALETTE の 2 つのイベントに対応する
- フレームウィンドウ・クラスのオブジェクトからビュークラスのオブジェクトにメッセージを転送する

```
//////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers

void CMainFrame::OnPaletteChanged(CWnd* pFocusWnd)
{
    CFrameWnd::OnPaletteChanged(pFocusWnd);

    CView* pView = GetActiveView();
    ASSERT(pView != NULL);

    SendMessageToDescendants(WM_PALETTECHANGED, (WPARAM)pView->m_hWnd);
}

BOOL CMainFrame::OnQueryNewPalette()
{
    CView* pView = GetActiveView();
    ASSERT(pView != NULL);

    pView->SendMessage(WM_QUERYNEWPALETTE, (WPARAM)pView->m_hWnd);
    return TRUE;
}
```

- 実際のパレット管理はビュークラスで行う

カラーパレット管理(2) (ビュークラスの処理)

- ClassWizard ではWM_PALETTECHAGED と WM_QUERYNEWPALETTE に対するイベント処理関数を登録できないので、ClassWizard がフレームウインドウ・クラスにメンバー関数のひな型を作ったのと同じ処理をハンド・コーディングで行う
  - CSimpleView.h を編集する

```
// Generated message map functions
protected:
    //{{AFX_MSG(CSimpleView)
    ...
    //}}AFX_MSG
    afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
    afx_msg BOOL OnQueryNewPalette();
    DECLARE_MESSAGE_MAP()
};
```

  - CSimpleView.cpp を編集する

```
BEGIN_MESSAGE_MAP(CGLEasyView, CView)
    //{{AFX_MSG_MAP(CGLEasyView)
    ...
    //}}AFX_MSG_MAP
    ON_WM_PALETTECHANGED()
    ON_WM_QUERYNEWPALETTE()
END_MESSAGE_MAP()
```

  - イベント処理関数を実装する

```
void CSimpleView::OnPaletteChanged(CWnd* pFocusWnd)
{
    if (pFocusWnd != this) {
        OnQueryNewPalette();
    }
}

BOOL CSimpleView::OnQueryNewPalette()
{
    if (m_pDC != NULL && m_pCPal != NULL)
    {
        CPalette* pOldPal = m_pDC->SelectPalette(m_pCPal, FALSE);
        UINT u = m_pDC->RealizePalette();
        if (u != 0) {
            InvalidateRect(NULL, TRUE);
            return u;
        }
    }
    return 0;
}
```

C.5 OpenGL による描画

OpenGL コンテキスト (OnCreate 関数)

- ピクセルフォーマット、カラーパレットの設定の後
- OpenGL レンダリング・コンテキストを作成
  - m_hrc =wglCreateContext(m_pDC->m_hDC);
- 必要に応じてカレントのレンダリング・コンテキストを設定する
  - wglMakeCurrent(hDC, m_hrc);
- wglMakeCurrent の使い方はアプリケーションによる

ビュークラスの描画関連コールバック関数 (1)

- 描画前にはカレントのレンダリング・コンテキストを設定する

```
wglMakeCurrent(m_pDC->m_hDC,  m_hrc);
```

ただしレンダリング・コンテキストの設定は重い処理なのでメモリー負荷がクリティカルでない場合は OnCreate 関数で一度設定してし、最後に OnDestroy 関数で解放し消去する

```
void CSimpleView::OnDestroy()
{
    CView::OnDestroy();
    wglMakeCurrent(NULL, NULL);
    if (m_hrc)
    {
        wglDeleteContext(m_hrc);
        m_hrc = NULL;
    }
}
```

- OnEraseBkgnd 関数
OpenGL レンダリングの前にクライアント・エリアを白で塗りつぶすのをやめるために、オーバーローディングして常に TRUE を返すようにする

```
BOOL CSimpleView::OnEraseBkgnd(CDC* pDC)
{
    return TRUE;
}
```

- OnInitialUpdate 関数
ディスプレイ・リストなど OpenGL プログラムの実行に当たって最初に一度だけ設定の必要なものをこの関数から呼ぶ

```
void CSimpleView::OnInitialUpdate()
{
    oglInitialize();

    CView::OnInitialUpdate();
}
```

ここで oglInitialize はディスプレイリストの構築、テクスチャの読み込みや設定を行う

ビュークラスの描画関連コールバック関数 (2)

- OnSize 関数
  ウィンドウ・サイズに変更があったときに呼び出される
  cx、cy には変更後のサイズがセットされる

```
void CSimpleView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);
    oglReshape(cx, cy);
}
```

  oglREshape で必要な処理は cx、cy に応じてビューポートを変更したり歪みが出ないように投影変換の再設定を行ったりする
  以下のコードはその一例である

```
void oglReshape(GLsizei w, GLsizei h)
{
    glViewport(0, 0, w,h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(30.0, (float)w/(float)h, 1.0, 10.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    polarView(5.0, 0.0, 0.0, 0.0);
}
```

- OnDraw 関数
  OpenGL のメインの描画関数を呼び出す
  ダブル・バッファリングのモードのときには、以下のコードのようにここでスワップバッファーする

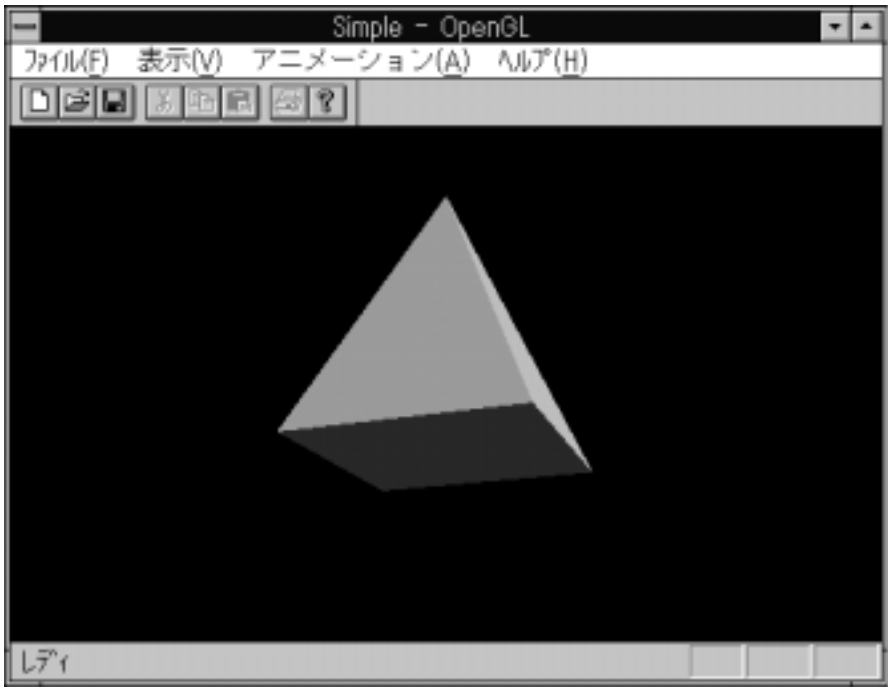
```
void CSimpleView::OnDraw(CDC* pDC)
{
    CSimpleDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // 次に oglDrawScene とあるのが OpenGL による描画関数
    oglDrawScene(m_rotx, m_roty) ;
    SwapBuffers(pDC->m_hDC) ;
```

```
}
```

C.6　アニメーション

アニメーション処理

- アニメーション処理はアプリケーションクラスとビュークラスのインターラクションが必要になる
  - アイドル時に呼び出されるコールバック関数 OnIdle はアプリケーションクラスのオブジェクトに属する
  - アプリケーションクラスのオブジェクトからビュークラスのオブジェクトにメッセージを送ってアニメーション表示させる
  - アプリケーションクラスのオブジェクトはビュークラスのオブジェクトのアドレスを自ら知る術がない
    一方、ビュークラスのオブジェクトは AfxGetApp 関数でアプリケーションクラスのオブジェクトのアドレスを調べられる
- 以下の処理でアニメーションを実現する
  - メニューなどからアニメーション・モードが設定された際に、ビュークラスのコールバック関数でアプリケーション・クラスのオブジェクトに自らのアドレス (this) を連絡して登録してもらう
  - アプリケーションクラスの OnIdle メンバー関数から登録されたビュークラスのオブジェクトにメッセージを送りアニメーション表示を実行する



アニメーション処理の実装例

- アニメーション処理に必要となる部分のコードを抜き出して示す
- アプリケーションクラスのヘッダー

```
public:
    ...
    CSimpleView* m_pIdleView;
    void SetIdleEvent(CSimpleView* pView)
        {m_pIdleView = pView;}
    ...
```

- アプリケーションクラスのメンバー関数

```
BOOL CSimpleApp::OnIdle(LONG lCount)
{
    if (!m_pIdleView) return FALSE;
    m_pIdleView->Tick();
    return TRUE;
}
```

- ビュークラスのヘッダー

```
public:
    void Tick(void); /* アニメーション表示 */
    void Spin(BOOL); /* アニメーションの On/Off */
    ...
protected:
    BOOL m_spin ;    /* アニメーションモードのフラグ */
    float m_rotx, m_roty; /* 回転角 */
    ...
protected:
    //{{AFX_MSG(CSimpleView)
    ...
    afx_msg void OnSpinStart();
    afx_msg void OnSpinStop();
    afx_msg void OnUpdateSpinStop(CCmdUI* pCmdUI);
    afx_msg void OnUpdateSpinStart(CCmdUI* pCmdUI);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

- ビュークラスのメンバー関数

```
void CSimpleView::Spin(BOOL bSpin)
{
    m_spin = bSpin ;
    // アプリケーションクラスのオブジェクトのアドレス
    CSimpleApp* pApp = (CSimpleApp*) AfxGetApp();

    pApp->SetIdleEvent(bSpin ? this : NULL);
}

// 以下の2つの関数はメニューからOnn/Offの伝達
void CSimpleView::OnSpinStart()
{
    Spin(TRUE);
}
void CSimpleView::OnSpinStop()
{
    Spin(FALSE);
}

// アプリケーションクラスのオブジェクトから
// 呼ばれるアニメーション表示関数
void CSimpleView::Tick(void)
{
    m_rotx += 10.0f;
    m_roty += 10.0f;
    if (m_rotx >= 360.0f) m_rotx = 0.0f ;
    if (m_roty >= 360.0f) m_roty = 0.0f ;
    Invalidate(FALSE) ;
    UpdateWindow();
}

// 以下の2つの関数はメニューのチェックのOn/Off用
void CSimpleView::OnUpdateSpinStop(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck((m_spin) ? 0 : 1);
}
void CSimpleView::OnUpdateSpinStart(CCmdUI* pCmdUI)
{
```

```
        pCmdUI->SetCheck((m_spin) ? 1 : 0);
    }
```

C.7　その他の話題

Windowsでのテクスチャマッピング

- OpenGL にはメモリー上のテクスチャデータの形式はあるが外部ファイルフォーマットはない
  NT のイメージファイルは DIB(BMP) ファイルが一般的
- Texture用のイメージファイルは、Intensity、Intensity + Alpha、RGB、RGB + Alpha の 4 種類がある
- Texture マッピングには Alpha 情報が重要
  例えば、街並みの「木」の表現

SGI RGBファイルの読み込みと書き込み

- SGI RGB ファイルのサポートライブラリ
  – libimage、ImageVision、Fstimage など
- Paul Haeberli の Fstimage ライブラリを移植
  – RGBA の配列の順番、プロトタイプ化、fopen を”rb” でなど
- IRIS 上で使用していたイメージファイルがそのまま使える
  http://www.webcity.co.jp/info/matumot/
- DIB の読み込みについては
  ”OpenGL V:Translating Windows DIBs”(http://www.microsoft.com/)

Windows 上の pros and cons

- よい点
  - Visual C++の統合環境
  - 安価な GUI 作成ツール (Visual C++/VisualStudio)
  - TrueType フォントを使った日本語 3 次元文字 (NT3.51 以降)
  - ActiveX や OLE などを利用できる
- 問題点
  - OpenGL 上のツール (IRIS Performer、ImageVision) がまだ不足
  - GLX のようなネットワーク透過な環境がない
  - OpenGL 専用 debuger がない

Internet 上の情報

- NetNews のニュースグループ
  comp.graphics.api.opengl
- 日本語のメーリングリスト
  - 次のようにメールを出すと自動登録される
    `To:OpenGL@caelum.co.jp`
    `Subject: subscribe`
    本文:プロフィール
  - OpenGL に興味がある方、現在使用/未使用を問わず参加可能
  - 問い合わせ：OpenGL-request@caelum.co.jp
- 英語のメーリングリスト
  - listserv@fatcity.com 宛に、本文に次のように書いて出す
    subject は何でもかまわない
    `sub OPENGL-GAMEDEV-L Your Name`
- ポインターは付属書の最後を参照

付 録D　Macintosh によるOpenGLプログラミング

本章の目的

Apple は 1999 年 1 月に SGI より OpenGL のライセンスを取得し、OpenGL を正式にサポートすることを発表した。Apple は Conix Graphics 社製の OpenGL for Macintosh の権利を技術者ごと買収し、完成度を高めたり、チューニング作業を行ってきた。その後、1999 年 5 月、MacOS 用 OpenGL の正式版がリリースされた。
この章では Macintosh での OpenGL アプリケーション開発に必用な情報をまとめて提供する。

- OpenGL のための Macintosh の環境設定方法を理解する
- MacOS 環境での OpenGL プログラミングを理解する
- DrawSprocket によるフルスクリーン・プログラムの作り方を理解する

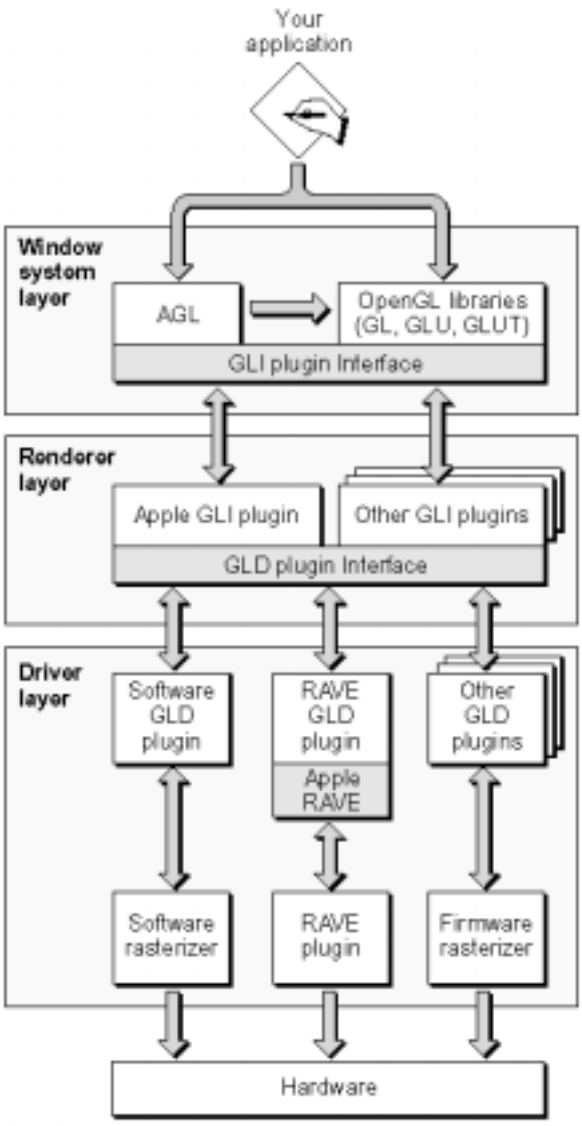
D.1 Macintosh OpenGLプログラミングの基礎

必要なハードウェア・リソース

- CPU は、PowerPC G3 以降の高速のものを選ぶ
- 古い PowerPC Mac でも一応動くが、動作環境として想定されていない
- 68k Mac では動作しない
- メイン・メモリーは CodeWarrior が快適に動作することを考慮する (OpenGL 動作の為には最低 32MB 以上のマシンが必用)
- ビデオ・カードは内蔵のもので動作し、特に新らたには必用としない
  - ATI の 3D Graphics チップの載った New G3 以降のマシンが推奨 (ATI RAGE-2, RAGE Pro, RAGE 128 など)
  - 機種によって、表示解像度との組み合わせで色数に制限がある場合があるので、考慮する
  - QuickDraw 3D RAVE 対応のビデオ・カードであれば、ある程度 OpenGL アクセラレーションされる

3次元グラフィックス・アクセラレーション・ボード

- Macintosh 用の専用 3 次元グラフィックス・アクセラレーション・ボードは少ない
  - Formac 社 (http://www.formac.com/) の ProFormance
  - 実売 259 ドル (3Dlabs 社製の Permedia 3 チップをベースとする)
  - 新しいグラフィックドライバが随時リリースされる
    http://www.formac.com/public/updates/beta/Graphics/
  - MacOS 上の OpenGL は下図のように処理される



必要なソフトウェア・リソース

- MacOS 9 以降の MacOS には Apple OpenGL 1.1.1 がバンドルされている
- iMac, iBook 等には MacOS 8.6 でも Apple OpenGL 1.1.1 があらかじめインストールされている。
- 1999 年 11 月に MacOS8.X, MacOS 9 用の Apple OpenGL 1.1.2 がリリースされた
- 1999-Novenber-15 版
- http://developer.apple.com/opengl/downloads.html
- さらに OpenGL SDK Version 1.0 を入手する (無料)
- http://developer.apple.com/opengl/downloads.html
- 開発環境 (コンパイラ) Metrowerks CodeWarrior Professional
  - CodeWarrior Professional 日本語版リリース 4 以降
  - 82,000 円 (税別) 実売 68000 円前後
  - CodeWarrior Professional 日本語版リリース 4 以降
  - アカデミック版:28,000 円 (税別)
  - http://www.metrowerks.co.jp/Products/Desktop/
- MacOS 8.0 以降、MacOS 8.1 以降を推奨、MacOS 9 も対応している
- OpenGL がバンドルされていない環境では別途 OpenGL Runtime を入手する
  - 1999-May-11 版
  - http://developer.apple.com/opengl/
  - SGI OpenGL Version 1.1 と同等 + 標準拡張 EXT + Apple 拡張
- インストール
  - QuickDraw 3D (RAVE) が Version 1.6 英語版に更新される
  - 現在一般に配布されている QuickDraw 3D ば英語版、日本語版とも Version 1.5.4 なので注意が必用
  - 日本語版 QuickDraw3D が既にインストールされている場合は二重にインストールされるので注意
- 拡張ライブラリ
  - OpenGL 実行ライブラリは、システムフォルダの拡張機能フォルダにインストールされる
  - いわゆるシェアードライブラリとして動作する
  - OpenGL の動作には QuickDraw 3D 関連のライブラリも同時に必用とする (QuickDraw 3D RAVE 等)
- Linux 環境で良く使われている OpenGL コンパチブルライブラリ Mesa の MacOS 版も存在するが、3Dfx 用のドライバが存在すること以外、特にメリットは無い
  - Mesa3dfxEngine 3.1b6
  - http://www.mesa3d.org/mac/

- Apple OpenGL の Conix OpenGL 1.5 との違い
  - ATI のドライバに対応している点
  - vertex array に対応している点
  - 各種動作速度チューニングがなされている点
  - Conix OpenGL ベースのアプリケーションは Apple OpenGL に差し換えると動作しない場合がほとんど

参考になる書籍

- CodeWarrior の日本語版にオンラインブックが付属している
  - Macintosh ではじめる C プログラミング
  - Macintosh ではじめる C++ プログラミング
  - Macintosh C プログラミング 第 2 版
  - 以上のものは Mac を使って C,C++言語を学ぶためのもの
- 進め！コードウォリア マックで始めるプログラミング
  - 仲川勝彦
  - ISBN 4-89563-884-7
  - 毎日コミュニケーションズ
  - 3200 円
- CodeWarrior ではじめる Macintosh プログラミング
  - トム・トンプソン Tom Thompson 著 武舎広幸
  - ISBN 4-931356-00-1
  - プレンティスホール出版／トッパン
  - 1994 年 12 月　B5 変形版　4800 円
- 基礎からはじめる Macitosh プログラミングテクニック
  - ダン P. サイドウ著　武舎 広幸＋久野 禎子訳
  - ISBN 4-931356-13-3
  - プレンティスホール出版
  - 1995 年 10 月発行 / 564 ページ / B5 変形版 / 4,369 円
- Inside Macintosh
  - Inside Macintosh は必ずしも購入の必用は無く、必用に応じて Web page を参照すれば良い
  - http://developer.apple.com/techpubs/macos8/mac8.html
- 雑誌 MacTech Magazine
  http://www.mactech.com/
  - OpenGL 特集号
  - 1998/12, vol.14 no.12.
  - 1999/01, vol.15 no.1.

Appleデベロッパープログラム

Apple が主催する開発者の為のプログラムがある。WWDC の招待券を含む高価なものから、開発者向けの情報、ソフトがつまった CD-ROM が定期的に送られてくる安価なものまでいくつかのレベルのものがある。(Premier 会員、Select 会員、Online 会員、Student 会員)

- http://developer.apple.com/ja/membership/

OpenGLプログラミング・スタイル

- GLUT、あるいはAUXを使った開発
  - UNIX、Windows、などへの移植性では最も優れる
  - ミニ・プログラムやプロトタイピング向きで、
    実用的プログラムを作るのには機能が不足
  - Mac の場合、標準入出力、コマンドラインのことを考慮しなければいけない
  - Apple OpenGL パッケージに含まれる GLUT は Version 2.3
- DrawSprocket を使った開発
  - メニューバー、ウィンドウ枠の表示されないフルスクリーンでの OpenGL 描画を行う
- AGL を使った開発
  - MacOS 専用のプログラムとなる
  - AGLContext を使ったプログラミング
  - GLM(OpenGL Memory library) を使いメモリ、ヒープのハンドリングを行う

D.2 GLUT を使ったプログラミング

GLUT を使ったプログラミング

- GLUT は AUX に含まれるイメージの読み込み機能はない
  – 例えば SGI イメージなら tkRGBImageLoad() を使う
- インクルードファイルは gl.h と glu.h は指定せず glut.h のみを指定する
  – gl.h, glu.h は glut.h の中で適切な順番で呼ばれる
  – 移植性を高めることができる
  "#ifdef MACOS"を使わなくていい
- 最低 3500KB の heap size と 256KB の stack size を確保する

ヘッダーとライブラリの追加

- ライブラリは以下のものを指定する
  - glut.lib
  - glut.rsrc
  - OpenGLLibraryStub
  - OpenGLUtilityStub
- 通常使うライブラリとして以下のものを指定する
  - MathLib
  - InterfaceLib
  - MSL C.PPC.Lib
  - MSL SIOUX.PPC.Lib
  - MSL RuntimePPC.Lib

D.3 DrawSprocketプログラミング

基本構造

- DrawSprocketプログラミングの流れ

```
main ( )
{
     InitGraf(...); ------ TookBox のイニシャライズ他
     HideCursor(); ---- カーソルの消去
     SetupScreen(...); --- DrawSprocket スクリーンのセットアップ
     setupAGL(...);  ----- AGL のセットアップ

        do {
                drawGL(...); ----- OpenGL の描画
        } while (!Button());

     cleanupAGL(...); ------ AGL の後処理
     ShutdownScreen(...); -- DrawSprocket スクリーンの後処理
     ShowCursor(); ------ カーソルの再描画
}
```

DrawSprocketプログラミングのポイント

- DrawSprocket.h と agl.h を include する
- フルスクリーン描画時は HideCursor()〜ShowCursor() でカーソルを隠す。
- DrawSprocket は Apple の提供するゲーム用 API 群 GameSprocket の一部であり、他にも InputSprocket, SoundSprocket 等がある。
- http://developer.apple.com/techpubs/mac/Sprockets/GameSprockets-85.html

OpenGL Java binding

- MacOS においても Java プログラミングで OpenGL API を使うことができる
- Magician
  http://www.arcana.co.uk/products/magician
- GL4Java for Mac
  http://members.home.net/gziemski/pages/projects/GL4Java.html
- JSparrow も Mac 版のリリースが予定されている
  http://www.pfu.co.jp/jsparrow/index-j.html

QuickDraw 3D からの移行

- OpenGL は固有のファイルフォーマットを持たないが、一方 QuickTime 3D は 3DMF という QuickDraw 3D と密着したバイナリファイルフォーマットが存在する。
- 資産として残った 3DMF フォーマットのファイルを OpenGL で活用することができる。
- 以下の 3DMF to OpenGL を使うと 3DMF の各モデル、サイズ、ユーザセッティング、テクスチャ等を OpenGL で使用する為に移行することができる。
- 3DMF to OpenGL Version 1.01 (ソースコードを含む)
  http://members.aol.com/ggadwa/3DMFOGL.sea.hqx

MacOSでのテクスチャマッピング

- OpenGL にはメモリー上のテクスチャデータの形式はあるが外部ファイルフォーマットはない
  MacOS のイメージファイルは PICT ファイルが一般的
- PICT ファイルフォーマットは最大横幅 4096 pixel の制限がある

MacOS 上の pros and cons

- よい点
  - CodeWarrior の統合環境
  - 安価な GUI 作成ツール (CodeWarrior PowerPlant)
  - 標準で、ある程度高速なビデオボードが載っているので、ターゲットが絞りやすい
- 問題点
  - OpenGL 上のツール (IRIS Performer、ImageVision) がまだ不足
    - その他の ToolKit を使う方法もある
    - RenderWare (キャノン販売 / 15 万〜65 万円)
    - QuickTime / QuickDraw 3D
  - GLX のようなネットワーク透過な環境がない
    - eXodus のような OpenGL に対応した X-Window client ソフトを使う方法もある
    - http://www.wpine.com
  - OpenGL 専用 debuger がない

Mac OpenGL プログラミング関連ツール

- NoMeMo Busters 1.7.3
  CodeWarrior のプロジェクト名などが化けるのを修正
  http://www.eac.co.jp/ ebata/soft/nomemo.html
- Mac CVS
  ソースコードバージョン管理ツールの MacOS 版
  http://www.maccvs.org/
- ResEdit 2.1.3
  Mac アプリケーションのリソースエディタ
  ftp://ftp.info.apple.com/Apple_Support_Area/
  Apple_Software_Updates/US/Macintosh/Utilities/ResEdit_2.1.3.sea.bin

Internet 上の Mac OpenGL 情報

- Apple Developer - OpenGL
  http://developer.apple.com/opengl/
- druid-'s GL Journal
  http://www.gamedev.net/opengl/
- ATI Technologies Inc.
  http://www.ati.com/
- News Group
  news:fj.sys.mac.programming
- OpenGL ディスカッションフォーラム
  http://www.opengl.org/Boards/discus/
- OpenGL for Mac Mailing List
  http://www.lists.apple.com/mac-opengl.html

  `majordomo@public.lists.apple.com` 宛に、本文に

  ```
  subscribe mac-opengl
  ```

  と書いたメールを送ってください。
  その後、登録確認のメールが返信されますので、
  それを送り返すようにしてください。

付 録E　ライブラリについて

本章では、本コースで利用する次の 2 つのライブラリについて述べます。

- liblaconia.a - 3 次元基本形状ライブラリ
- librgbImageFile.a - SGI-rgb イメージファイルライブラリ

E.1　3 次元基本形状ライブラリ (liblaconia.a)

liblaconia.a

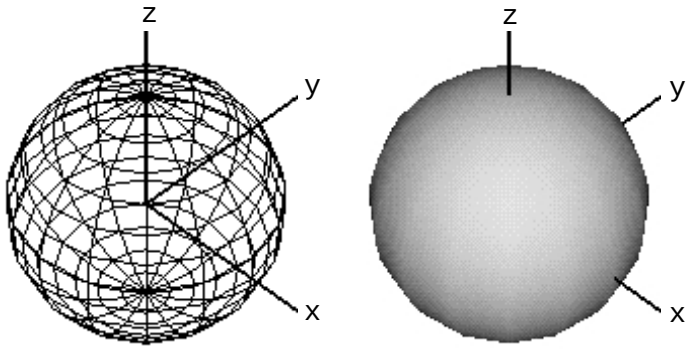
- インクルードファイルは laconia.h
- 基本的な 9 種類の 3 次元形状の描画ルーチンが入っている
- 光源処理に使われる法線ベクトルはルーチン内で指定される
- 色やテクスチャー座標値は外部から操作できる
- 内部でディスプレイリストを使っていない
  – 外部でディスプレイリスト化できる

球

void lacSphere(GLdouble radius, char type)

- radius は球の半径
- type はソリッド/ワイヤーフレームの指定
  – 'w'（ワイヤーフレーム)、's'（ソリッド）のいずれか

lacSphere(10.0, 'w');

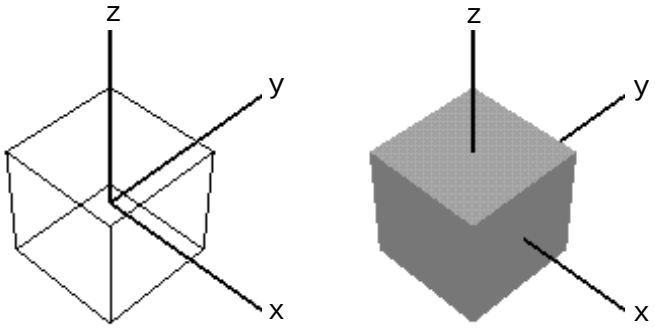


立方体

void lacCube(GLdouble size, char type)

- size は 1 辺の長さ

lacCube(10.0, 'w');

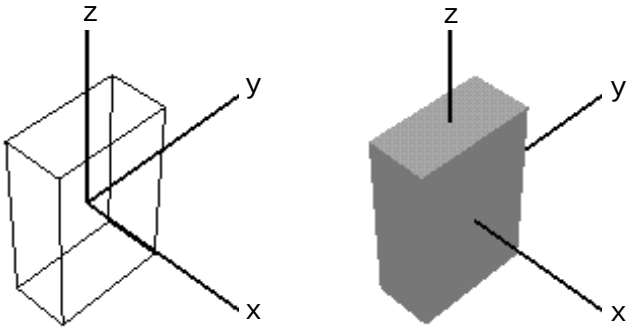


座標原点を中心とする直方体

void lacBox(GLdouble xSize, GLdouble ySize, GLdouble zSize, char type)

- xSize、ySize、zSize はそれぞれ x、y、z 軸方向の長さ

lacBox(5.0, 10.0, 15.0, 'w');

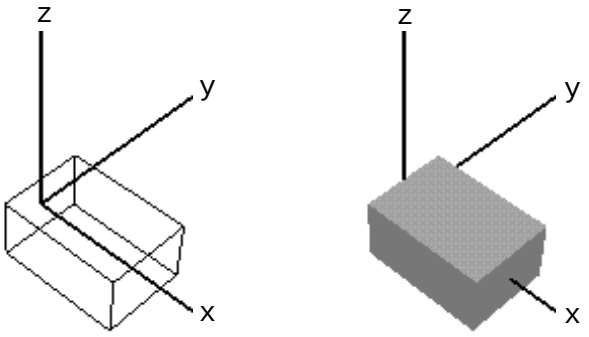


ロボットアーム用の直方体

void lacArm(GLdouble xSize, GLdouble ySize, GLdouble zSize, char type)

- xSize、ySize、zSize はそれぞれ x、y、z 軸方向の長さ

lacArm(10.0, 7.0, 5.0, 'w');

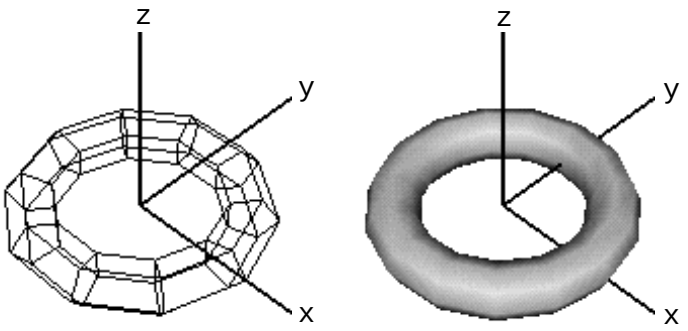


円環体（ドーナツ型）

void lacTorus(GLdouble innerRadius, GLdouble outerRadius, char type)

- innerRadius はパイプの太さ（半径）
- outerRadius は座標原点からパイプ中心までの長さ

lacTorus(2.0 ,8.0 ,'w');

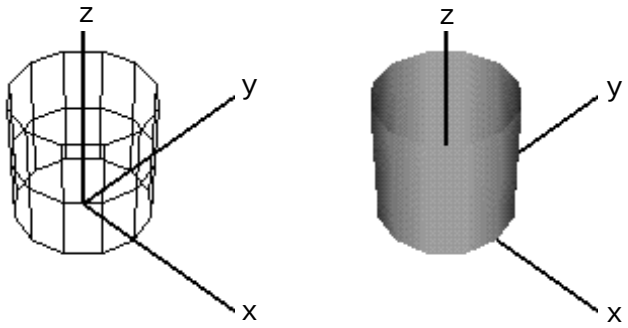


円柱

void lacCylinder(GLdouble radius, GLdouble height, char type)

- radius は底面半径
- height は高さ

lacCylinder(5.0, 10.0, 'w');

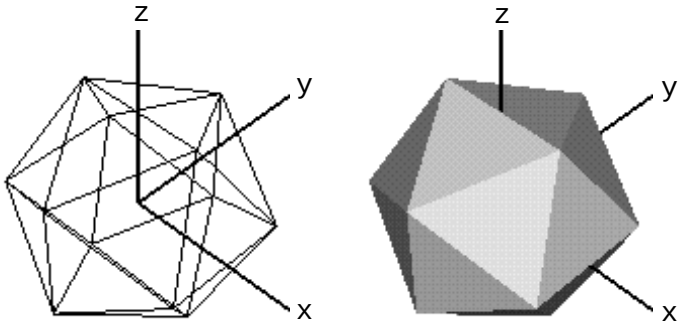


20 面体

void lacIcosahedron(GLdouble radius, char type)

- radius は半径（座標原点から頂点までの距離）

lacIcosahedron(10.0, 'w');

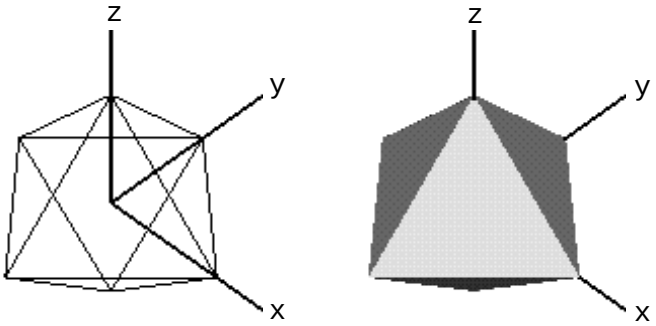


8面体

void lacOctahedron(GLdouble radius, char type)

- radius は半径（座標原点から頂点までの距離）

lacOctahedron(10.0, 'w');

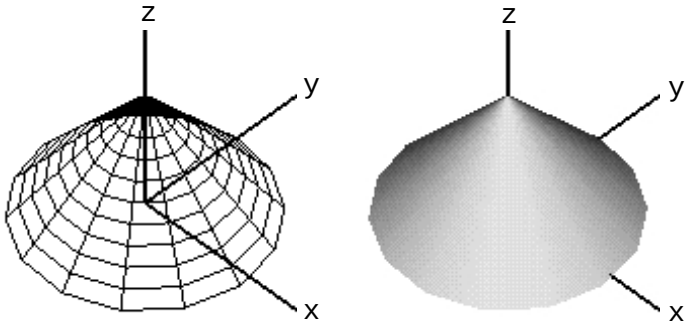


円錐

void lacCone(GLdouble base, GLdouble height, char type)

- base は底面半径
- height は高さ

lacCone(10.0, 10.0, 'w');



E.2 SGI-rgb イメージファイルライブラリ (libsgiimage.a)

libsgiimage.a

- インクルードファイルは sgiimage.h

イメージファイルの読み込み

unsigned int *rgbReadImageFile(char *name, int *pwidth, int *pheight)

- name はファイル名
- pwidth、pheight ポインターの指す領域にイメージの幅と高さを返す
- 返り値はイメージ配列の先頭アドレス

イメージファイルの作成

int rgbWriteImageFile(char *name, int width, int height, unsigned int *parray)

- name はファイル名
- width、height はイメージの幅と高さ
- parray は rgb イメージ配列の先頭アドレス
- 書き込みが成功したときは 1 を返し、失敗したときには 0 を返す

付 録F　OpenGLのネーミング規約

OpenGLの関数名、定数名、タイプ名

- OpenGL の関数名
  - OpenGL の関数は各々次の prefix が付く
    - コアライブラリ gl
    - ユーティリティ・ライブラリ glu
    - インターフェイス・ライブラリ glX、wgl、pgl、agl
    - AUX ライブラリ aux
    - glut ライブラリ glut
  - 単語の切れ目は大文字で示す
  - postfix で引数の個数、型を示す
  - 接尾子で次元と精度、列挙か配列かを区別することができる
    - 2 次元か 3 次元、あるいは 4 次元（同次座標系）
    - 精度の識別は次の表による
      glVertex2i(1, 3);、glVertex2f(1.0f, 3.0f);
    - 配列の先頭アドレスを指定する場合は末尾に v が付く
      double vert[2] = {1.0, 3.0};
      glVertex2dv(vert);

接尾子	精度	C 言語のタイプ	OpenGL のタイプ
b	8-bit 整数	signed char	GLbyte
s	16-bit 整数	short	GLshort
i	32-bit 整数	integer	GLint, GLsizei
f	32-bit 浮動小数点数	float	GLfloat, GLclampf
d	64-bit 浮動小数点数	double	GLdouble, GLclampd
ub	8-bit 非負整数	unsigned char	GLubyte, GLboolean
us	16-bit 非負整数	unsigned short	GLushort
ui	32-bit 非負整数	unsigned int	GLuint, GLenum, GLbitfield

- OpenGL の定数名
  - 全て大文字で単語間は下線（_）で区切る
  - 例えば、GL_COLOR_BUFFER_BIT、GL_RGBA
- OpenGL のタイプ名
  - GL で始まりあとは小文字で表す
  - 例えば、GLenum、GLfloat、GLvoid

OpenGL情報

- OpenGL 必見 WEB ページ
  - The OpenGL WEB Site (はじめにアクセスすべきページ)
    `http://www.opengl.org/`
  - MJK(Mark J Kilgard) と Jon Leech による、GLUT ソフトウェア、包括的なポインター集、プログラミングテクニック集、技術文書、X Journal の記事など
    (特に tips の方は OpenGL プログラマー必見)
    `http://reality.sgi.com/opengl/`
    `http://reality.sgi.com/opengl/tips/`
  - SGI OpenGL WWW Center (OpenGL の仕様書など)
    `http://www.sgi.com/Technology/OpenGL/`
  - OpenGL のよくある質問とその解答集、日本独自の補遺版あり
    `http://www.nk-exa.co.jp/~andoh/opengl/faq/`（日本語版）
    `http://www.sgi.com/Technology/openGL/faq.html`　（英語版）
- NetNews では次のグループに情報がある

  `news:comp.graphics.api.opengl`

- OpenGL メーリングリスト関連情報
  - 日本の OpenGL メーリングリスト
    メーリングリストの概要と登録方法は次のホームページ参照
    `http://www.caelum.co.jp/~gougi/`
    過去の記事のアーカイブ (管理 安藤幸央)
    `http://www.egroups.co.jp/group/opengl-jp`

  - ゲーム開発者のための OpenGL メーリングリスト (英語)
    listserv@fatcity.com 宛てに、Subject は空欄で、本文へ次のコマンドを書いてメイルを出す
    `sub OPENGL-GAMEDEV-L ローマ字であなたの名前`

- PC の 3D ハードウェアの総合情報

  `http://www.star.rcast.u-tokyo.ac.jp/~yanagida/pc3d-card.html`

- 日本の OpenGL サイト
  - 松本の OpenGL_Japan 関連ホームページ
    (OpenGL_Japan の講演集、OpenGL ActiveX コントロールなど各種例題集他)
    `http://www.nk-exa.co.jp/~matumot/`

  - KGT の OpenGL_Japan ホームページ
    http://www.kgt.co.jp/opengl/
  - 合木 茂氏のホームページ (筒口氏による GLUT 日本語訳マニュアルあり)
    http://www.caelum.co.jp/~gougi/
  - 寺西 忠勝氏の OpenGL 入門講座と、OpenGL 製品の機能調査ツール集とその結果集
    http://www.asahi-net.or.jp/~yw3t-trns/
  - 寺田 健彦氏の OpenGL チューニング情報 (SGI 各種マシン対応)
    http://reality.sgi.com/terada_nsg/
- OpenGL 各種開発ツール
  - GLUT for UNIX, Windows
    http://reality.sgi.com/opengl/
  - the Mesa 3-D graphics library
    http://www.mesa3d.org/
  - FLTK (ポータブルな GUI 作成ツール)
    http://fltk.easysw.com/
  - XForms (X 用 GUI 作成ツール)
    http://bragg.phys.uwm.edu/xforms/
- ベンチマーク情報
  - The OpenGL Performance Characterization Project
    http://www.specbench.org/gpc/
  - Indy3D ベンチマークテスト (Sense8/Mitsubishi)
    http://www.sense8.com/indy3d/
  - GLAZE ベンチマークテスト (Evans and Sutherland)
    http://www.es.com/
  - Yocc 氏による日本製 OpenGL ベンチマーク
    http://www.bekkoame.or.jp/~yoccy/
- その他の情報
  - O2 などを使った OepnGL とビデオ混合プログラミング情報
    http://reality.sgi.com/cpirazzi_esd/lg/index.html
  - OpenGL/X window System 混合プログラミング情報
    (Overlay プレーン、ダイアルボタンボックスの使い方など)
    http://www.nk-exa.co.jp/~bobby/

OpenGLプログラミング

1995年9月1日　　初　版　第1刷発行
1997年5月15日　第2版　第1刷発行
1997年9月12日　第2版　第2刷発行
2000年3月31日　第3版　オンラインバージョン (PDF 発行)

著者　　安藤 幸央
　　浦城 康弘
　　松本 昌幸
　　山中 政彦
　　吉田 和弘
　　山腰 洋一郎
表紙　　山中 政彦
制作　　(株) エヌ・ケー・エクサ / マルチメディア技術マーケティング部
発行　　(株) エヌ・ケー・エクサ / 科学技術システム本部

本書はIRIX上のpLaTeX2ϵで作成しました。