

- 5 typedef unsigned int GLenum
- 5 typedef unsigned int GLbitfield
- 5 typedef int GLint
- 5 typedef int GLsizei
- 5 typedef unsigned int GLuint
- 5 typedef float GLfloat
- 5 typedef float GLclampf
- 5 typedef double GLdouble
- 5 typedef void GLvoid
- 5 void glutInit(int \*argc, char \*\*argv)
  - 5 GLUTライブラリを初期化します。
  - 5 「argc」と「argv」はmain関数の引数、すなわちコマンドライン引数を渡します。これらの引数は、コマンドラインのオプション指定時に用いられます。
- 5 void glutInitDisplayMode(unsigned int mode)
  - 5 ディスプレイの表示モードを設定します。
  - 5 「glutInitDisplayMode(GLUT\_RGBA|GLUT\_DOUBLE|GLUT\_DEPTH)」のように書くと、「RGBAカラーモデル」で「ダブルバッファ」を使い、「デプスバッファ」も使うという指定になります。
- 5 void glutInitWindowSize(int width, int height)
  - 5 ウィンドウの初期サイズを設定します。
  - 5 「width」はウィンドウの幅、「height」はウィンドウの高さになります。
- 5 void glutInitWindowPosition(int x, int y)
  - 5 ウィンドウの左上の位置を指定する。引数は共にピクセル値。
- 5 int glutCreateWindow(char \*title)
  - 5 ウィンドウを生成する。引数はそのウィンドウの名前となる。
- 5 void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)
  - 5 「glClear(GL\_COLOR\_BUFFER\_BIT)」でウィンドウを塗りつぶす際の色を指定します。
  - 5 「red」「green」「blue」はそれぞれ「赤」「緑」「青色」の成分の強さを示すGLclampf型(float型と等価)の値で、0~1の間の値をもちます。1が最も明るく、この3つに(0,0,0)を指定すれば「黒色」になり、(1,1,1)を指定すれば「白色」になります。
  - 5 最後の「alpha」は「 $\alpha$ 値」と呼ばれ、OpenGLでは不透明度として扱われます(0で透明、1で不透明)。ここではとりあえず「1」にしておいてください。
- 5 void glutMainLoop(void)
  - 5 GLUTのイベントが発生するまで、待機状態になります。
- 5 void glutSwapBuffers(void)
  - 5 描画の最後で記述する。この関数が実行されると、バックバッファの内容がフロントバッファに転送される。
- 5 void glClear(GLbitfield mask)
  - 5 「mask」に指定したバッファのビットを初期化します。
  - 5 「glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT)」と指定すると「カラーバッファ」と「Zバッファ」が初期化されます。
- 5 void glEnable(GLenum cap)
  - 5 GLenum型の引数「cap」に指定した機能を使用可能にします。
  - 5 「glEnable(GL\_DEPTH\_TEST)」を実行すると、それ以降「Zバッファ」を使います。
  - 5 「glEnable(GL\_LIGHTING)」を実行すると、それ以降「陰影付け」の計算をします。
  - 5 「glEnable(GL\_LIGHT0)」を実行すると、0番目の光源を点灯します。
  - 5 「glEnable(GL\_TEXTURE\_2D)」を実行すると、それ以降「テクスチャマッピング」ができるようになります。
- 5 void glDisable(GLenum cap)
  - 5 引数「cap」に指定した機能を使用不可にします。

- ❏ `void glutDisplayFunc(void (*)(void))`
  - ❏ 引数は開いたウィンドウ内に描画する関数へのポインタです。ウィンドウが開かれたり、他のウィンドウによって隠されたウィンドウが再び現われたりしてウィンドウを再描画する必要があるときに、この関数が実行されます。したがって、この関数内で図形表示を行います。
- ❏ `void glutReshapeFunc(void (*)(int width, int height))`
  - ❏ 引数には、ウィンドウがリサイズされたときに実行する関数のポインタを与えます。この関数の引数にはリサイズ後のウィンドウの幅と高さが渡されます。
- ❏ `void glutKeyboardFunc(void (*)(unsigned char key, int x, int y))`
  - ❏ 引数には、キーがタイプされたときに実行する関数のポインタを与えます。この関数の引数「key」には、タイプされたキーのASCIIコードが渡されます。また、「x」と「y」にはキーがタイプされたときのマウスの位置が渡されます。
- ❏ `void glutMouseFunc(void (*)(int button, int state, int x, int y))`
  - ❏ 引数には、マウス・ボタンが押されたときに実行する関数のポインタを与えます。この関数の引数「button」には、押されたボタン (GLUT\_LEFT\_BUTTON, GLUT\_MIDDLE\_BUTTON, GLUT\_RIGHT\_BUTTON) が渡されます。引数「state」には、「押した」(GLUT\_DOWN) のか「離れた」(GLUT\_UP) のかが渡されます。また、引数「x」と「y」にはその位置が渡されます。
- ❏ `void glutMotionFunc(void (*)(int x, int y))`
  - ❏ 引数には、マウスのいずれかのボタンを押しながらマウスを動かしたときに実行する関数のポインタを与えます。この関数の引数「x」と「y」には、現在のマウスの位置が渡されます。
  - ❏ この設定を解除するには、引数に「0」(ヌル・ポインタ)を指定します (stdio.hなどの中で定義されている記号定数「NULL」を使ってもかまいません)。
- ❏ `void glutPostRedisplay(void)`
  - ❏ ウィンドウを再描画します。glutDisplayFunc()で登録したコールバック関数が呼び出されます。
- ❏ `void glViewport(GLint x, GLint y, GLsizei width, GLsizei height)`
  - ❏ 「ビューポート」を設定します。「ビューポート」とは、開いたウィンドウの中で、実際に描画される領域のことをいいます。正規化デバイス座標系の2点(-1,-1), (1,1)を結ぶ線分を対角線とする矩形領域がここに表示されます。最初の2つのGLint型 (int型と等価)の引数「x,y」にはその領域の左下隅の位置、後の2つのGLsizei型 (int型と等価)の「width」と「height」には、それぞれ幅と高さをデバイス座標系の値、すなわちディスプレイ上の画素数で指定します。glutReshapeFuncで指定されたコールバック関数の引数「width,height」にはそれぞれウィンドウの幅と高さが入っていますから、glViewport(0,0,width,height)はリサイズ後のウィンドウの全面を表示領域に使うことになります。

- 5 void glMatrixMode(GLenum mode)
  - 5 設定する変換行列を指定します。引数「mode」が「GL\_MODELVIEW」なら「モデルビュー変換行列」を指定し、「GL\_PROJECTION」なら「透視変換行列」を指定します。
- 5 void glLoadIdentity(void)
  - 5 これは変換行列を初期化します。座標変換の合成は行列の積で表されますから、この関数を使って変換行列に初期値として単位行列を設定します。
- 5 void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar)
  - 5 変換行列に透視変換の行列を乗じます。
  - 5 最初の引数「fovy」はカメラの画角であり、「度」で表します。これが大きいほど広角(透視が強くなり、絵が小さくなります)になり、小さいほど望遠レンズになります。
  - 5 2つ目の引数「aspect」は画面のアスペクト比(縦横比)であり、「1」であればビューポートに表示される図形のx方向とy方向のスケールが等しくなります。
  - 5 3つ目の引数「zNear」と4つ目の引数「zFar」は表示する奥行き方向の範囲で、「zNear」は手前(前方面)、zFarは後方(後方面)の位置を示します。この間にある図形が描画されます。
- 5 void glMultMatrixd(GLdouble \*m)
  - 5 16個の値からなる行列mを現在の行列と乗算し、その結果を現在の行列にします。
  - 5 16個の値は以下のような4×4行列に対応しています。
 
$$\begin{pmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{pmatrix}$$
- 5 void glPushMatrix(void)
  - 5 glMatrixMode()で指定している現在の変換行列を保存します。
- 5 void glPopMatrix(void)
  - 5 glPopMatrix()で保存した変換行列を復帰します。したがって、「glPushMatrix()」を呼び出した後、glTranslated()やglRotated()あるいはgluLookAt()などを使って変換行列を変更しても、「glPopMatrix()」を呼び出すことによって、それ以前の変換行列に戻すことができます。
- 5 void glTranslated(GLdouble x, GLdouble y, GLdouble z)
  - 5 変換行列に平行移動の行列を乗じます。引数はいずれもGLdouble型で、3つの引数「x」「y」「z」には現在の位置からの相対的な移動量を指定します。
- 5 void glRotated(GLdouble angle, GLdouble x, GLdouble y, GLdouble z)
  - 5 変換行列に回転の行列を乗じます。引数はいずれもGLdouble型で、1つ目の引数「angle」は回転角、残りの3つの引数「x」「y」「z」は回転軸の方向ベクトルです。
  - 5 回転角「angle」の単位は度(°)(degree)です。C言語のcos関数などの引数の単位はラジアンなので注意してください。

- 5 void glBegin(GLenum mode)  
void glEnd(void)
  - 5 「glBegin」と「glEnd」の間に指定した頂点座標を使って、描画を行います。
  - 5 描画内容は「mode」に指定します。「mode」には「GL\_LINE\_STRIP」や「GL\_LINES」や「GL\_TRIANGLES」や「GL\_QUADS」や「GL\_POLYGON」などが指定できます。
- 5 void glVertex3d(GLdouble x, GLdouble y, GLdouble z)  
void glVertex3dv(GLdouble \*v)
  - 5 3次元の座標値を設定します。
  - 5 引数はGLdouble型の(x, y, z)で指定します。
  - 5 引数として3つの要素をもつGLdouble型の配列vで指定する関数も用意されています。
- 5 void glNormal3d(GLdouble nx, GLdouble ny, GLdouble nz)  
void glNormal3dv(GLdouble \*v)
  - 5 単位法線ベクトルを設定します。
  - 5 引数はGLdouble型の(nx, ny, nz), または, 3つの要素をもつGLdouble型の配列vで指定します
- 5 void glTexCoord2d(GLdouble s, GLdouble t)  
void glTexCoord2dv(GLdouble \*v)
  - 5 2次元のテクスチャ座標を設定します。
  - 5 引数はGLdouble型の(s, t), または, 2つの要素を持つGLdouble型の配列vで指定します。
- 5 void glColor3d(GLdouble red, GLdouble green, GLdouble blue)
  - 5 これから描画するものの色を指定します。
  - 5 引数の型はGLdouble型で, 「red」「green」「blue」にはそれぞれ「赤」「緑」「青」の強さを「0~1」の範囲で指定します。
- 5 void glMaterialfv(GLenum face, GLenum pname, GLfloat \*params)
  - 5 表面属性を定義する。
  - 5 「face」に「GL\_FRONT」を指定すると, ポリゴンの表面のみに属性を設定します。
  - 5 「pname」に「GL\_DIFFUSE」を指定すると, 「params」でfloat型の配列を指定することで, 材質の拡散RGBA値を設定できます。その際はglMaterialfvを使います。

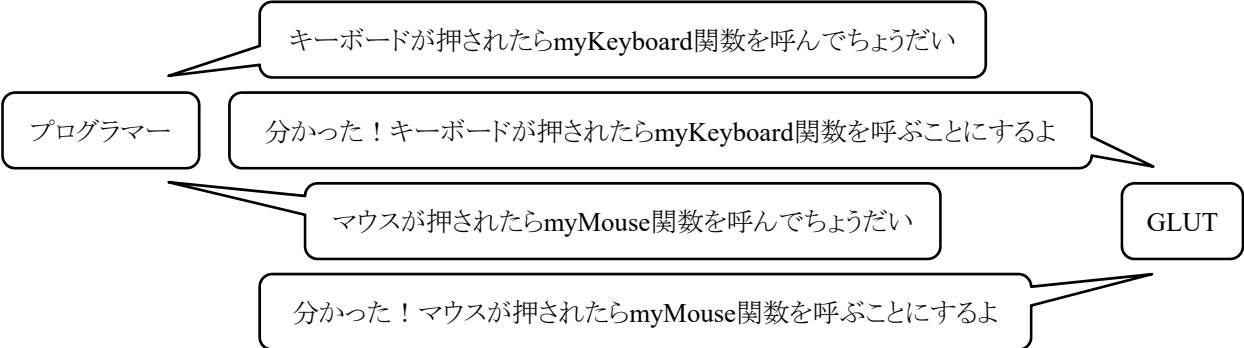
## OpenGL

- ❏ `void glGenTextures(GLsizei n, GLuint *textures)`
  - ❏ テクスチャオブジェクトの名称を取得します
  - ❏ `glGenTextures(1, &texName)`とすることで, `unsigned int texName`に, テクスチャのID番号が格納されます
  - ❏ この`texName`は`glBindTexture`関数や`glDeleteTextures`関数を呼び出す際に必要です
- ❏ `void glDeleteTextures(GLsizei n, GLuint *textures)`
  - ❏ テクスチャオブジェクトを削除します
- ❏ `void glBindTexture(GLenum target, GLuint texture)`
  - ❏ 指定したテクスチャオブジェクトを有効化します
  - ❏ `target`には`GL_TEXTURE_2D`を指定します
    - ❏ `texture`には`glGenTextures`関数で取得したテクスチャ番号を指定します
    - ❏ `texture`に0を指定した場合はテクスチャオブジェクトの使用を停止します
- ❏ `void glTexEnvf(GLenum target, GLenum pname, GLfloat param)`
  - ❏ テクスチャ関数を設定します
  - ❏ `target`に`GL_TEXTURE_ENV`を指定し, `pname`に`GL_TEXTURE_ENV_MODE`を指定します
    - ❏ このとき, `param`に`GL_DECAL`を指定すると, テクスチャをシールのように貼り付けるようになります. 物体の材質やライトによる陰影の影響を受けません
    - ❏ このとき, `param`に`GL_MODULATE`を指定すると, テクスチャは物体の色と混合されます. 物体の材質やライトによる陰影の影響を受けます
- ❏ `void glTexParameterf(GLenum target, GLenum pname, GLfloat param)`
  - ❏ テクスチャを表示する際の各処理方法を制御するパラメータを指定します
  - ❏ `target`には`GL_TEXTURE_2D`を指定します
  - ❏ この授業では, `pname`に`GL_TEXTURE_WRAP_S`を指定して, `param`に`GL_CLAMP`を指定して呼び出します
  - ❏ この授業では, `pname`に`GL_TEXTURE_WRAP_T`を指定して, `param`に`GL_CLAMP`を指定して呼び出します
  - ❏ この授業では, `pname`に`GL_TEXTURE_MAG_FILTER`を指定して, `param`に`GL_LINEAR`を指定して呼び出します
  - ❏ この授業では, `pname`に`GL_TEXTURE_MIN_FILTER`を指定して, `param`に`GL_LINEAR`を指定して呼び出します
- ❏ `void glTexImage2D(GLenum target, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, GLvoid *pixels)`
  - ❏ 2次元テクスチャのデータを設定します
  - ❏ `target`は`GL_TEXTURE_2D`を指定します
  - ❏ `level`は0を指定します
  - ❏ `internalFormat`は`GL_RGB`を指定します
    - ❏  $\alpha$ 値も利用する場合は`GL_RGBA`を指定します
  - ❏ `width`と`height`は画像の幅と高さを指定します. 画像の1辺は2の累乗でなければいけません.
  - ❏ `border`は0を指定します
  - ❏ `format`は`GL_RGB`を指定します
    - ❏  $\alpha$ 値も利用する場合は`GL_RGBA`を指定します
  - ❏ `type`は`GL_UNSIGNED_BYTE`を指定します
  - ❏ `pixels`に画像データの配列を指定します

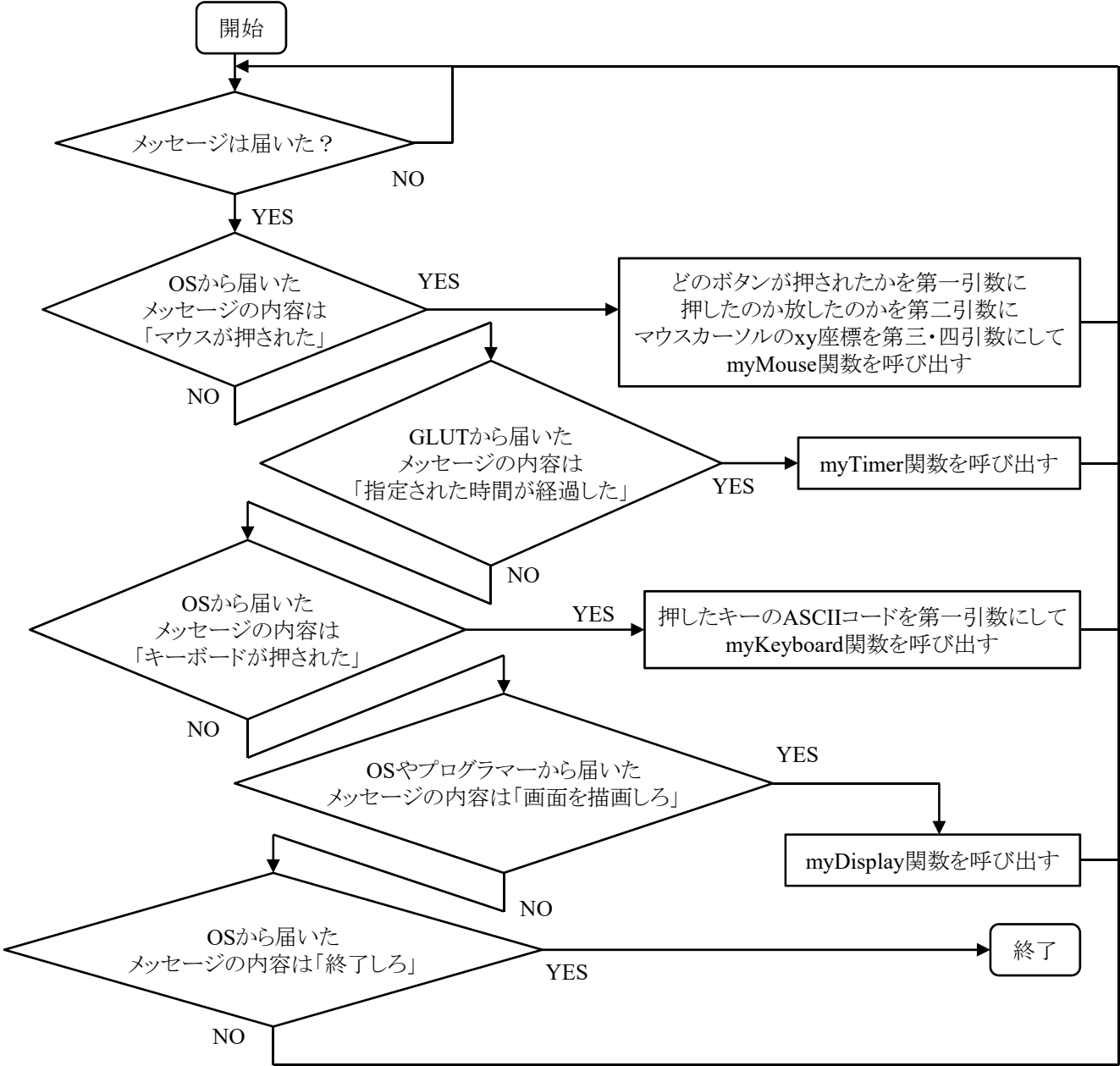
# イベントドリブン型プログラミング

- glutMainLoopは無限ループしているだけで、メッセージが届くのをひたすら待ち続けます
- メッセージを受け取ったら、メッセージに応じた処理をして、また再び無限に待機します

## コールバック関数の指定



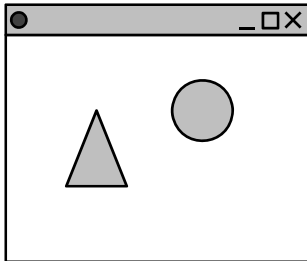
## ループ中の動作の例



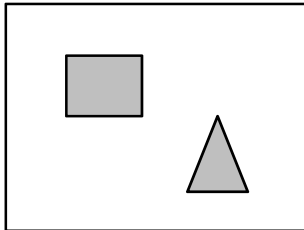
## 画面の描画

🔗 `glClear`で画面を消去して`glutSwapBuffers`で描画します

## 描画の流れ

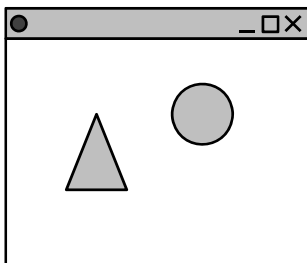


ウィンドウ表示用のバッファ



メモリ内にあるバッファ

```
glClear(GL_COLOR_BUFFER_BIT);  
画面消去
```

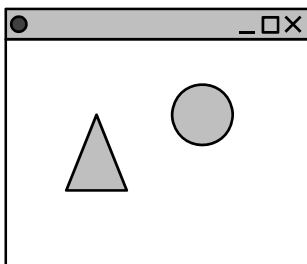


ウィンドウ表示用のバッファ

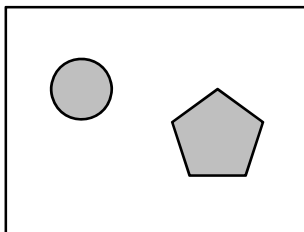


メモリ内にあるバッファ

```
glBegin~glEnd  
図形の描画
```

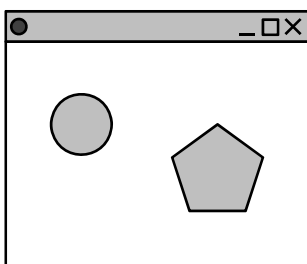


ウィンドウ表示用のバッファ

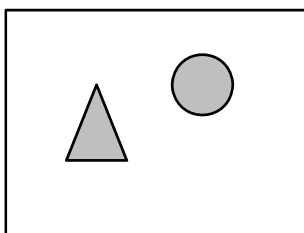


メモリ内にあるバッファ

```
glutSwapBuffers();  
ウィンドウに表示
```



ウィンドウ表示用のバッファ



メモリ内にあるバッファ

```
void ディスプレイコールバック関数()  
{  
    // 画面消去  
    glClear(GL_COLOR_BUFFER_BIT);  
  
    // 図形の描画  
    glBegin(GL_QUADS);  
    glVertex3d(-1.0, -1.0, 0.0);  
    glVertex3d(1.0, -1.0, 0.0);  
    glVertex3d(1.0, 1.0, 0.0);  
    glVertex3d(-1.0, 1.0, 0.0);  
    glEnd();  
  
    // ウィンドウに表示  
    glutSwapBuffers();  
}
```

## 点, 線, ポリゴンの描画

点, 線, ポリゴンを描くのに, 次のようにglBegin()とglEnd()および, glVertex\*()を用いる.

```
glBegin(mode);  
  glVertex*(p0);  
  glVertex*(p1);  
  .....  
  glVertex*(pn);  
glEnd();
```

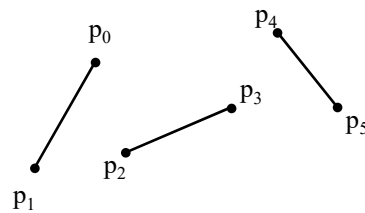
modeは描く図形の種類を指定し, p0, p1, ..., pnは座標位置を意味する.

## modeの種類

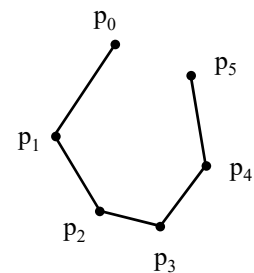
- GL\_LINE\_STRIP 最初の頂点から最後の頂点まで線分を連結して描画する.
- GL\_LINES 二つの頂点を結んだ直線を生成する.
- GL\_TRIANGLES 三つ一組の頂点を, それぞれ独立した三角形として描画する.
- GL\_QUADS 四つ一組の頂点を, それぞれ独立した四角形として描画する.
- GL\_POLYGON 単独の凸ポリゴンを描画する.

## ポリゴン描写の注意点

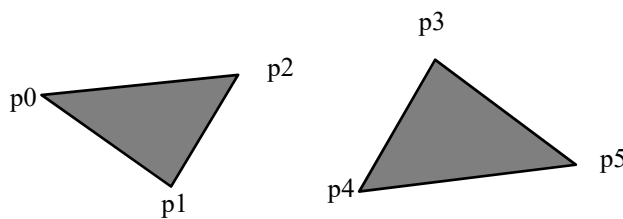
- ポリゴン (polygon) とは多角形の意味
- 頂点座標は反時計回りに設定すること
  - ポリゴンの表面と裏面を区別するため
  - 視点から見て頂点座標が反時計回りに配置されているポリゴンは表面, 時計回りの場合は裏面と約束されている



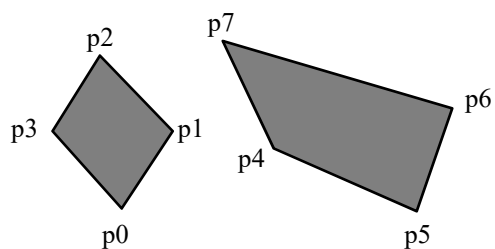
GL\_LINES



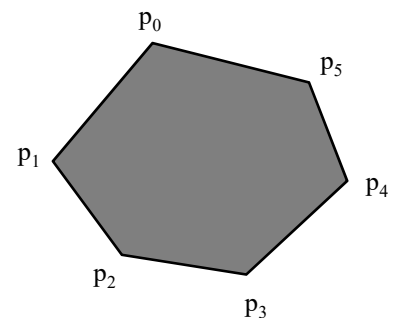
GL\_LINE\_STRIP



GL\_TRIANGLES



GL\_QUADS

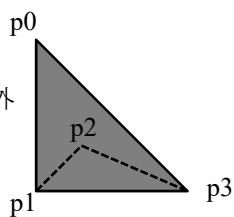
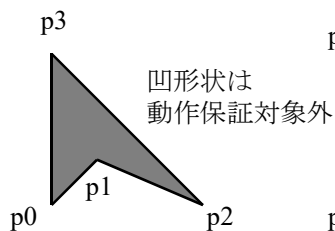
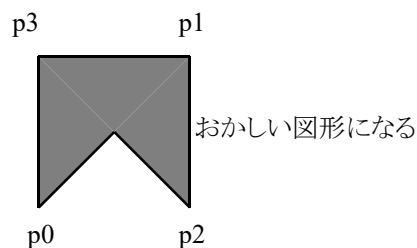
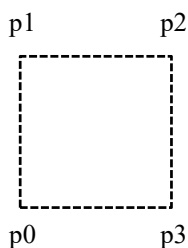
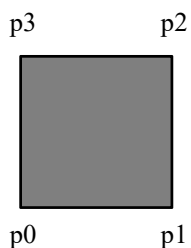


GL\_POLYGON

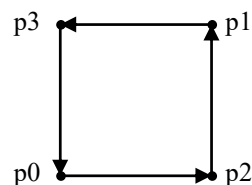
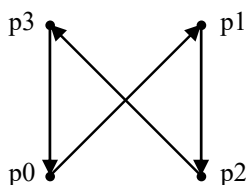


## 四角形の頂点の順番

4つの頂点の順番によって表示される図形が異なる



右の例は、 $p_0 \rightarrow p_1 \rightarrow p_2 \rightarrow p_3$ の順番だからおかしくなるのであって、 $p_0 \rightarrow p_2 \rightarrow p_1 \rightarrow p_3$ の順番で `glVertex?d`関数を呼び出せば、正しく描画される

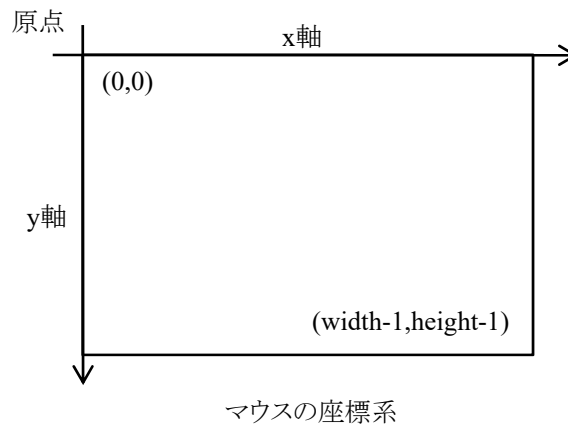
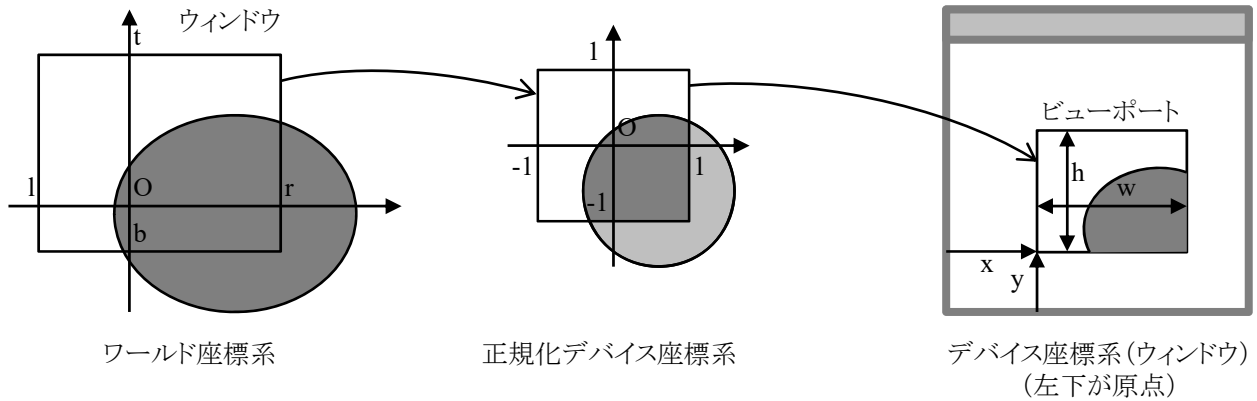


OpenGLの仕様とは異なる使い方をした場合(この場合、凸多角形にしか対応していないという仕様を無視して、非凸多角形を描画しようとした場合)の動作は保証されない。

<http://monobook.org/wiki/OpenGL>

通常、凹多角形は三角形に分割して、凸多角形で表現するのが普通(検索キーワード: 三角形分割)

## ウィンドウとビューポート



## マウス

- マウスのボタンを押したか離れたか、そのときの画素位置は `glutMouseFunc` で指定するコールバック関数で確かめる
- マウスをドラッグした画素位置は `glutMotionFunc` で指定するコールバック関数で確かめる

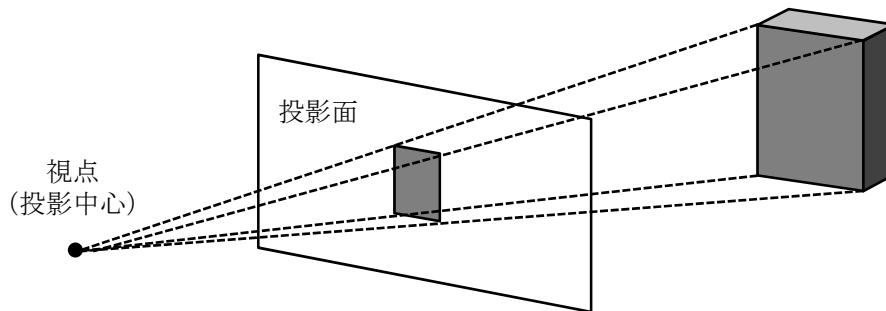
コールバック関数の設定  
`glutMouseFunc(myMouse);`  
`glutMotionFunc(myMotion);`

```
void myMouse(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        printf("左ボタンが押された位置は (%d,%d)\n", x, y);
    }
    if (button == GLUT_LEFT_BUTTON && state == GLUT_UP) {
        printf("左ボタンが放された位置は (%d,%d)\n", x, y);
    }
}

void myMotion(int x, int y)
{
    printf("現在のマウスカーソルの位置は (%d,%d)\n", x, y);
}
```

## 透視投影 (perspective projection)

- 📌 視点と投影面を指定
- 📌 人間の見え方に近い



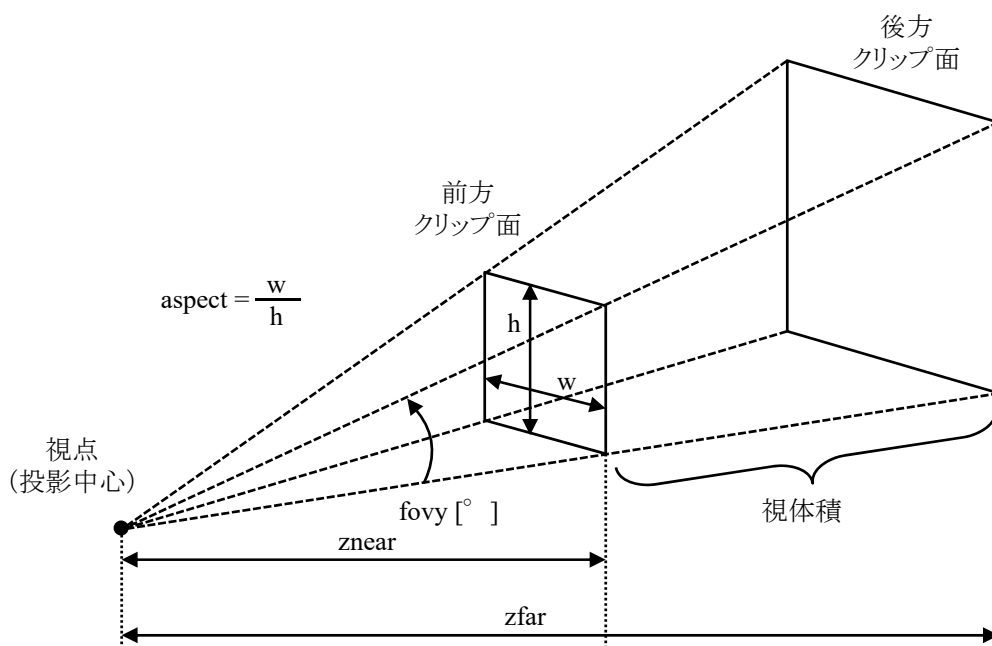
## 視体積

- 📌 オブジェクトが見える領域を視体積(view volume)という
- 📌 この錐台の外にあるオブジェクトは表示されない

## 関数

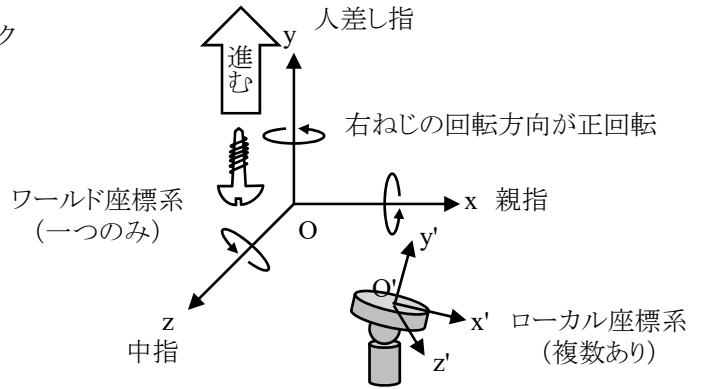
- 📌 `void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar)`
  - 📌 変換行列に透視変換の行列を乗じます.
  - 📌 最初の引数「fovy」はカメラの画角であり、「度」で表します. これが大きいくほど広角(透視が強くなり, 絵が小さくなります)になり, 小さいほど望遠レンズになります.
  - 📌 2つ目の引数「aspect」は画面のアスペクト比(縦横比)であり, 「1」であればビューポートに表示される図形のx方向とy方向のスケールが等しくなります.
  - 📌 3つ目の引数「zNear」と4つ目の引数「zFar」は表示する奥行き方向の範囲で, 「zNear」は手前(前方面), zFarは後方(後方面)の位置を示します. この間にある図形が描画されます.

## 透視投影の視体積



## 右手系

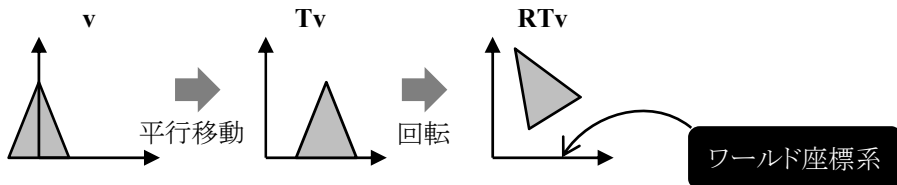
- ワールド座標系: ユーザが描く図形やオブジェクト全てに対して共通な座標系である
- ローカル座標系: 各オブジェクト固有の座標系であり, ワールド座標系の中に複数存在する



## モデリング変換

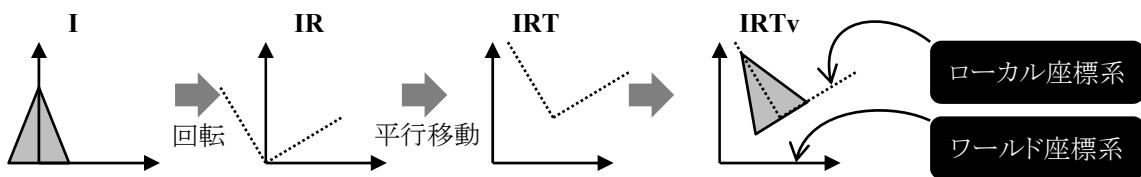
- OpenGLは, オブジェクトの平行移動(translation), 回転(rotation), スケーリング(scaling)の三つのモデリング変換を用意している
- OpenGLは, 2次元や3次元の頂点を扱うが, 内部では統一的に4次元ベクトル(x, y, z, w)を用いる. これを同次座標という.
- OpenGLでは, 点 $v=(x\ y\ z\ 1)^T$ を点 $v'=(x'\ y'\ z'\ 1)^T$ に変換する一般式は $v'=Tv$ で表される
- $T$ は $4 \times 4$ の行列で, 変換行列を表す
- 変換行列には, モデルビュー行列(model view matrix)と投影行列(projection matrix)がある
- OpenGLでは, ローカル座標系の考え方をすると, プログラムの処理順序に合う

### ワールド座標系の考え方



```
glLoadIdentity();
glRotate*();
glTranslate*();
glWireCone();
```

### ローカル座標系の考え方



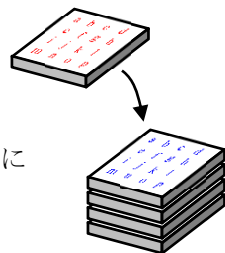
## スタック

- OpenGLでは, 変換行列をスタック領域に格納することができる

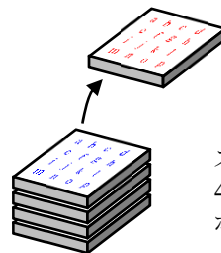
PUSH (プッシュ)

POP (ポップ)

スタック(積み重ねたもの)に  
4×4行列を  
プッシュ(押し込む)する



スタック(積み重ねたもの)から  
4×4行列を  
ポップ(引き出す)する



## 平行移動

### glVertex?d関数(頂点の座標を変更する場合)

📌 glVertex?d関数で頂点の座標を指定できるので、この数値を変えることで図形を移動できます

例1

```
glBegin(GL_QUADS);  
glVertex3d(posx-0.5, posy-0.5, 0.1);  
glVertex3d(posx+0.5, posy-0.5, 0.1);  
glVertex3d(posx+0.5, posy+0.5, 0.1);  
glVertex3d(posx-0.5, posy+0.5, 0.1);  
glEnd();
```

### glTranslated関数(座標系を変更する場合)

📌 glTranslated関数でローカル座標系を平行移動できるので、ローカル座標系を移動させたあと、図形の描画命令を書けばいいわけです

📌 ローカル座標系から見れば図形は移動していませんが、ワールド座標系から見れば図形が移動しているかのように見えます

例1

```
glPushMatrix();  
glTranslated(posx, posy, 0.5);  
glutSolidSphere(1.0, 10, 10);  
glPopMatrix();
```

例2

```
glPushMatrix();  
glTranslated(posx, posy, 1.9);  
myHuman(timestep, cycle);  
glPopMatrix();
```

例3

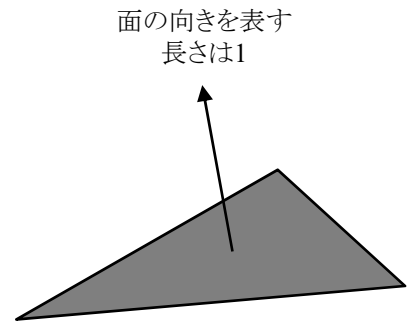
```
glPushMatrix();  
glTranslated(posx, posy, 0.1);  
glBegin(GL_QUADS);  
glVertex3d(-0.5, -0.5, 0.0);  
glVertex3d(+0.5, -0.5, 0.0);  
glVertex3d(+0.5, +0.5, 0.0);  
glVertex3d(-0.5, +0.5, 0.0);  
glEnd();  
glPopMatrix();
```

## 法線

- 法線は、面に垂直な方向を指しているベクトルです。OpenGLでは、各ポリゴン、または各頂点に対して法線を指定できます。
- 現在の法線は`glNormal*()`で設定します。それ以降に呼び出された`glVertex*()`で指定した頂点に現在の法線が割り当てられます。

## 用例

```
glBegin(GL_POLYGON);  
  glNormal3d(n0x, n0y, n0z);  
  glVertex3d(v0x, v0y, v0z);  
  glNormal3d(n1x, n1y, n1z);  
  glVertex3d(v1x, v1y, v1z);  
  glNormal3d(n2x, n2y, n2z);  
  glVertex3d(v2x, v2y, v2z);  
  glNormal3d(n3x, n3y, n3z);  
  glVertex3d(v3x, v3y, v3z);  
glEnd();
```

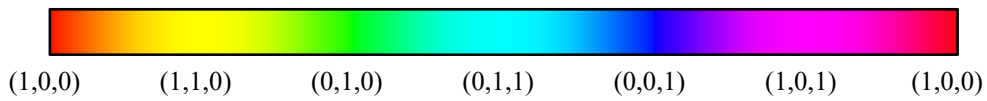
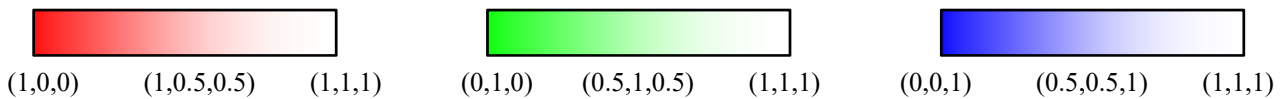
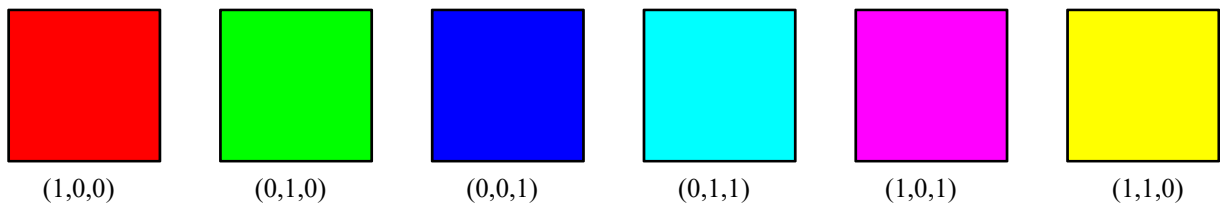
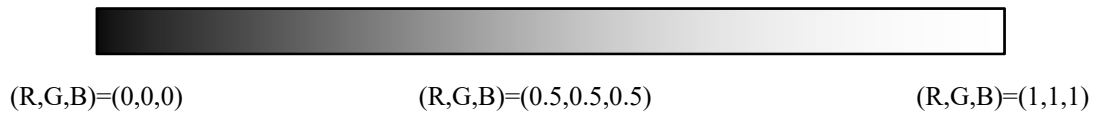
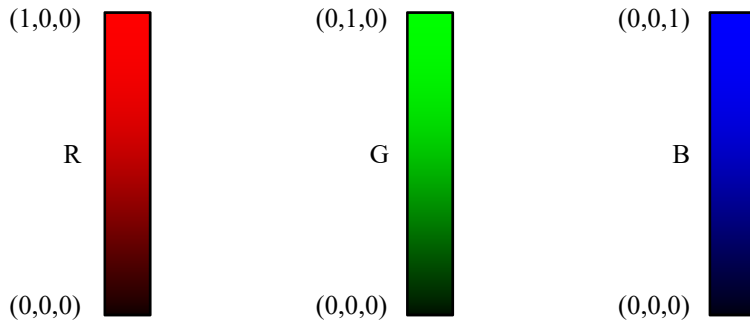


## 関数の説明

- `void glNormal3d(GLdouble nx, GLdouble ny, GLdouble nz);`
  - 法線ベクトルを設定します。

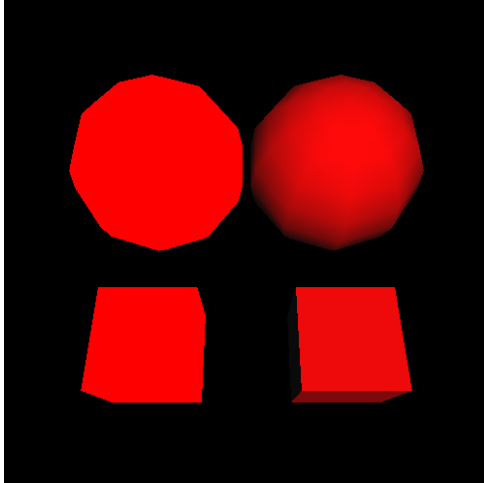
## 色の表現

- 光の三原色で表す
- R: Red 赤, G: Green 緑, B: Blue 青
- glClearColor関数で背景色を指定, glColor3d関数で描画する図形の色を指定します
- OpenGLのこれらの関数の引数は, 浮動小数点の0~1の値で表します
  - ちなみに, 画面や画像ファイルなどの画素は通常, RGBそれぞれ8bitずつ, 合計24bitで表します. 整数で0~255の値で表します. OpenGLは0~1ですのでご注意ください.
- 0が暗くて, 1が明るいです



## 図形の色

←ライティングがオフの場合



←ライティングがオンの場合

自分が今作っているプログラムはライティングがオンの状態ですか？オフの状態ですか？ソースコードをよく理解して適切な方法で色を設定してください。

## glColor3d関数(ライティングしない場合)

- ❏ 初期状態ではglDisable(GL\_LIGHTING), すなわち, ライティング計算をしない状態になっています。
- ❏ 陰影は計算されません。
- ❏ 図形の色はglColor3d関数で指定します。

例1

```
glColor3d(1.0, 0.0, 0.0);  
glutSolidSphere(1.0, 10, 10);
```

例2

```
glColor3d(1.0, 0.0, 0.0);  
glBegin(GL_QUADS);  
glVertex3d(-1.0, -1.0, 0.0);  
glVertex3d(1.0, -1.0, 0.0);  
glVertex3d(1.0, 1.0, 0.0);  
glVertex3d(-1.0, 1.0, 0.0);  
glEnd();
```

## glMaterialfv関数(ライティングする場合)

- ❏ glEnable(GL\_LIGHTING)で, ライティング計算をする状態になります。
- ❏ glEnable(GL\_LIGHT0)などで, 光源を点灯する必要があります。
- ❏ 光源と法線の関係を使って陰影が計算されます。
- ❏ 図形には法線が正しく設定されていなければいけません。
- ❏ 図形の色はglMaterialfv関数で指定します。

例1

```
float red[4] = { 1.0f, 0.0f, 0.0f, 1.0f };  
glEnable(GL_LIGHTING);  
glEnable(GL_LIGHT0);  
glMaterialfv(GL_FRONT, GL_DIFFUSE, red);  
glutSolidSphere(1.0, 10, 10);
```

例2

```
float red[4] = { 1.0f, 0.0f, 0.0f, 1.0f };  
glEnable(GL_LIGHTING);  
glEnable(GL_LIGHT0);  
glMaterialfv(GL_FRONT, GL_DIFFUSE, red);  
glBegin(GL_QUADS);  
glNormal3d(0.0, 0.0, 1.0);  
glVertex3d(-1.0, -1.0, 0.0);  
glVertex3d(1.0, -1.0, 0.0);  
glVertex3d(1.0, 1.0, 0.0);  
glVertex3d(-1.0, 1.0, 0.0);  
glEnd();
```



## 光源の有効・無効化

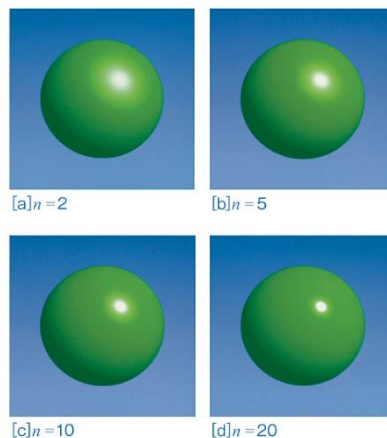
- ☞ `glEnable(GL_LIGHTING);`
    - ☞ 光源による陰影を計算する機能をオンにする
  - ☞ `glEnable(GL_LIGHTn);`
    - ☞ 番号`n=0~7`の光源を点灯する
  - ☞ `glLightfv(GL_LIGHTn, pname, params);`
    - ☞ 番号`n=0~7`の光源の属性を設定する
  - ☞ `glMaterialfv(GL_FRONT, pname, params);`
    - ☞ 光源が点灯された状態で描画したいシーンのOpenGL命令を実行する
  - ☞ `glDisable(GL_LIGHTn);`
    - ☞ 番号`n=0~7`の光源を消灯する
  - ☞ `glDisable(GL_LIGHTING);`
    - ☞ 光源による陰影を計算する機能をオフにする
- ☞ 光源0番はデフォルトで白色光が設定されているが、光源1番以降はデフォルトで真っ暗になっているので、光源1番以降を使用する際は光源の明るさの設定を忘れずに

## 物体表面の材質の設定

- ❏ `void glMaterialfv(GLenum face, GLenum pname, GLfloat *param)`
- `void glMaterialf(GLenum face, GLenum pname, GLfloat param)`
  - ❏ 表面属性を定義する.
  - ❏ `face`に`GL_FRONT`を指定すると、ポリゴンの表面のみに属性を設定します.
  - ❏ `param`でfloat型の数値を指定する場合は`glMaterialf`を使い、`param`でfloat型の配列を指定する場合は`glMaterialfv`を使う

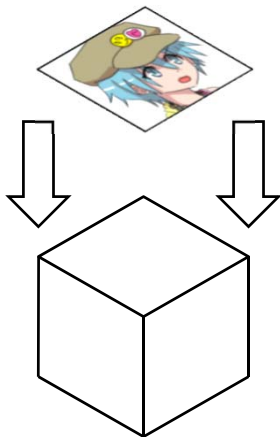
pname	params	パラメータ
GL_AMBIENT	float型の4次元配列	材質の環境RGBA値
GL_DIFFUSE	float型の4次元配列	材質の拡散RGBA値
GL_SPECULAR	float型の4次元配列	材質の鏡面RGBA値
GL_SHININESS	float型のスカラー値	鏡面係数 0以上の値を指定する 小さい数値を指定すると鏡面反射が広がり、表面が粗い材質を表現できる 大きい数値を指定すると鏡面反射がせばまり、表面が滑らかな材質を表現できる

■ 図 4.33——フォンの反射モデルによるハイライトの違い

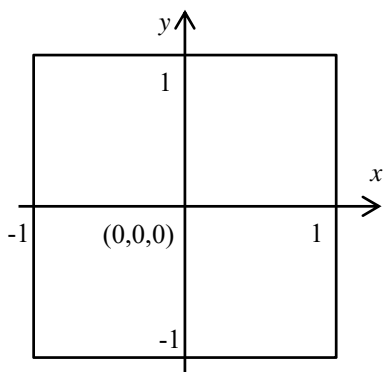
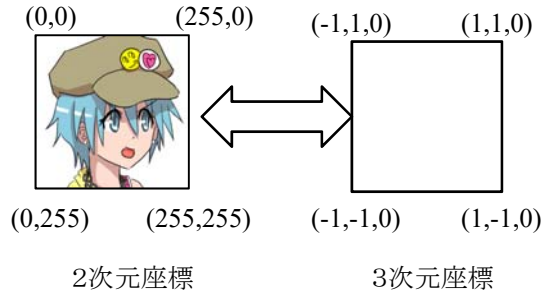


テクスチャマッピングの座標系

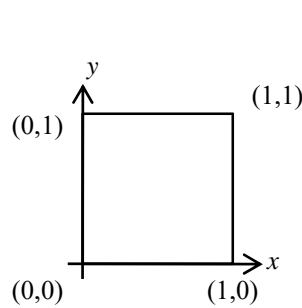
3次元の面に2次元の画像を貼り付ける=テクスチャマッピング



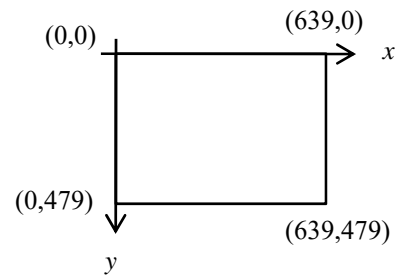
点の座標を対応づける



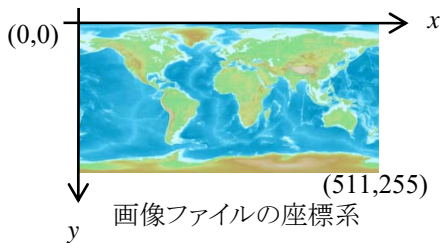
OpenGLの空間の座標系(初期状態)



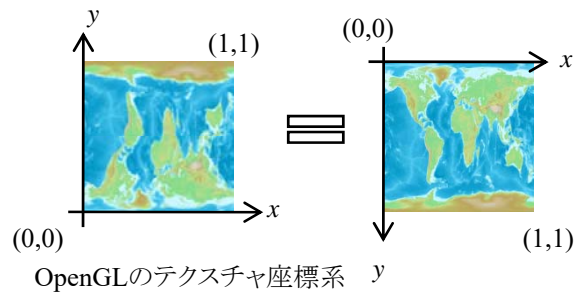
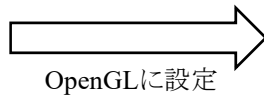
OpenGLのテクスチャ座標系



画像やディスプレイの座標系



画像ファイルの座標系



OpenGLのテクスチャ座標系

```
glBegin(GL_QUADS);
glTexCoord2d(0.0, 0.0); glVertex3d(-1.0, 1.0, 0.0);
glTexCoord2d(0.0, 1.0); glVertex3d(-1.0, -1.0, 0.0);
glTexCoord2d(1.0, 1.0); glVertex3d(1.0, -1.0, 0.0);
glTexCoord2d(1.0, 0.0); glVertex3d(1.0, 1.0, 0.0);
glEnd();
```

## テクスチャマッピングのやり方

```
unsigned int texName;
glGenTextures(1, &texName);
glBindTexture(GL_TEXTURE_2D, texName);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, imagew, imageh, 0, GL_RGB, GL_UNSIGNED_BYTE, imagedata);
glEnable(GL_TEXTURE_2D);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

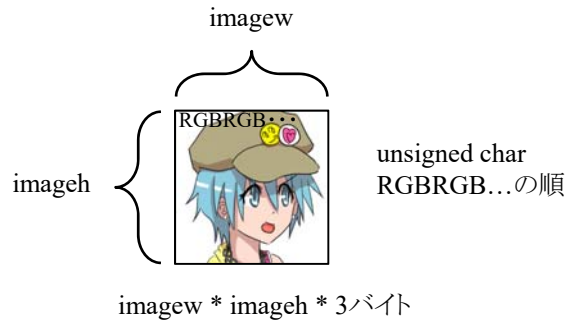
// glClearを呼んだあと、描画したいシーンの命令を実行

glBindTexture(GL_TEXTURE_2D, 0);
glDeleteTextures(1, &texName);
glDisable(GL_TEXTURE_2D);

glutSwapBuffers();
```

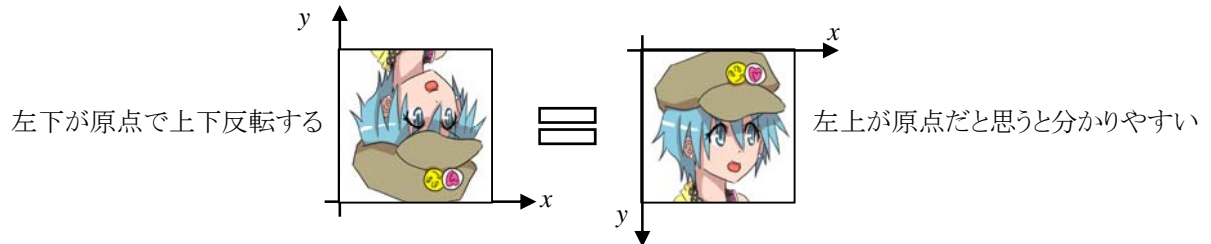
「おまじない」  
詳しく知りたい人は教科書等で調べてみよう！

imagew, imageh, imagedata以外は「おまじない」  
imagew: 画像の幅 [pixel]  
imageh: 画像の高さ [pixel]  
imagedata: 画像データ



画像の幅と高さは $64 \times 64$ 以上で、それぞれの辺が $2^n$ の値でなければいけない  
貼れる画像の最大サイズはハードウェアによって異なる

1つ前のページで説明した通り、OpenGLのテクスチャ座標系はこんな感じ

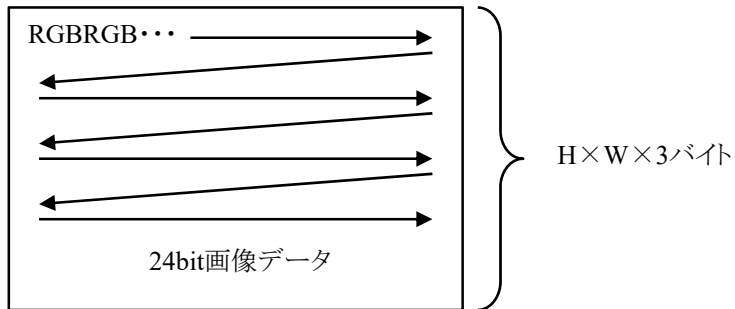


OpenGLの座標系は左下が原点なのだからimagedataは本当は左下から右上に並べるべきだが  
このように、左上から右下に並べたほうが分かりやすい

glVertex3d関数で頂点の座標を指定する前に、glTexCoord2dでテクスチャ座標を指定すればOK

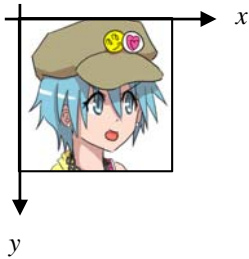
画像ファイル

- ✖ 通常、各画素のRGB値それぞれ1バイトで表現する(画素の明るさを0~255で表現)
- ✖ 左上の画素から右下に向かってスキャンラインオーダーで並べた配列データで表現できる
- ✖ 高さH, 幅Wの画像ならH×W×3バイトの配列で表現できる

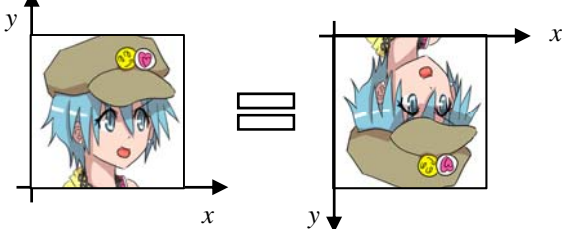


- ✖ このままファイルとして表現してしまうと、高さHと幅Wが分からない
- ✖ 高さHや幅Wやその他の情報をヘッダとして、ファイルの先頭部分に格納する
- ✖ 画像ファイルの種類は様々なものが存在するが、それぞれ、ヘッダの内容や形式、画像データの表現形式、などに違いがある
- ✖ Windowsでは標準でBMPファイルが利用されている
- ✖ BMP(ビー・एम・ピー)はbitmap(ビットマップ)の略で、拡張子はbmpである
- ✖ 圧縮形式と非圧縮形式があるが、圧縮形式は詳細が不明であるので扱わないほうがいい
- ✖ 1bit2色, 4bit16色, 8bit256色の形式もあるが、ここでは24bit1670万色の形式のみを説明する
- ✖ 画像ファイルの多くは左上を原点とすることが多いが、BMPファイルは左下を原点とする珍しいファイル形式である
- ✖ 画像ファイルの多くはRGBRGB...の順に輝度値を並べることが多いが、BMPファイルはBGRBGR...の順に並べる珍しいファイル形式である
- ✖ 画像データは行単位で、1行のバイト数が4の倍数でない場合は、右端に詰め物(padding)が入られる

ディスプレイ画面の座標系  
多くの画像ファイルの座標系



BMPファイルの座標系



ヘッダ

BMPファイル識別文字B	1 byte
BMPファイル識別文字M	1 byte
ファイルサイズ [byte]	4 byte
予約領域1 [byte]	2 byte
予約領域2 [byte]	2 byte
ヘッダサイズ [byte]	4 byte
情報ヘッダサイズ [byte]	4 byte
画像xサイズ [dot]	4 byte
画像yサイズ [dot]	4 byte
面数	2 byte
色ビット数 [bit/dot]	2 byte
圧縮方式	4 byte
圧縮サイズ [byte]	4 byte
水平解像度 [dot/m]	4 byte
垂直解像度 [dot/m]	4 byte
色数	4 byte
重要色数	4 byte

14+40=54バイト

BGRBGR...
24bit画像データ

x×y×3バイト

BMPファイルはリトルエンディアン  
(Intelなどが採用. 0x1234という値を, 0x34を1バイト目に, 0x12を2バイト目に格納する方式)

## テクスチャ画像



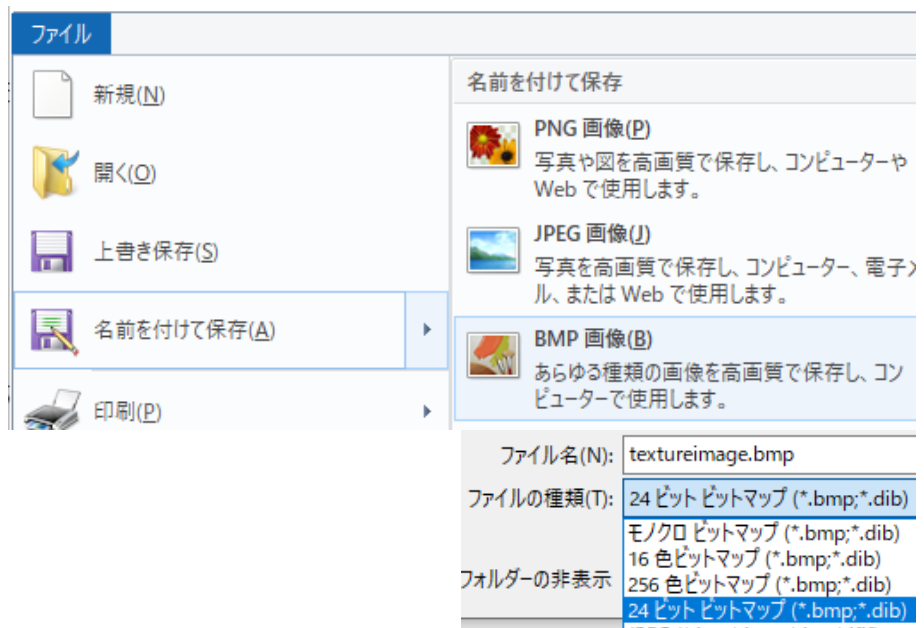
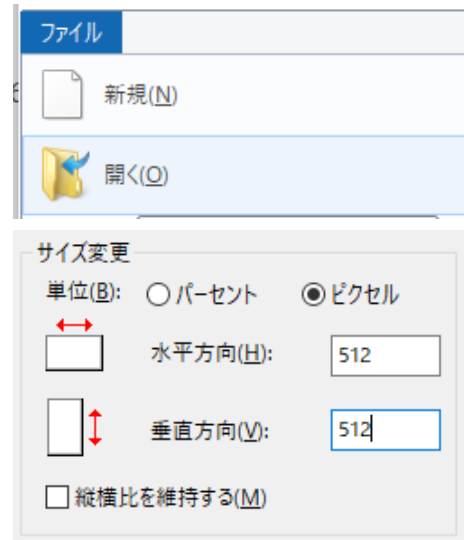
画像の高さと幅は64, 128, 256, 512, 1024など

## bmpファイル

- ☞ サンプルソースコードで対応しているファイル形式はbmpファイルのみ

## ファイル変換

- ☞ Windowsメニューで[Windowsアクセサリ]-[ペイント]をクリック
- ☞ [ファイル]-[開く]でファイルを開く
- ☞ [ホーム]-[イメージ]-[サイズ変更]をクリック
  - ☞ [縦横比を維持する]のチェックを外す
  - ☞ [単位]を[ピクセル]にする
  - ☞ [水平方向]と[垂直方向]にサイズを入力する
- ☞ [ファイル]-[名前を付けて保存]-[BMP画像]をクリック
- ☞ [ファイルの種類]で[24ビットビットマップ]を選ぶ



# バーチャルトラックボール

```
コールバック関数の設定  
glutMouseFunc(myMouse);  
glutMotionFunc(myMotion);
```

anglex = angley = 0.0

moving = false

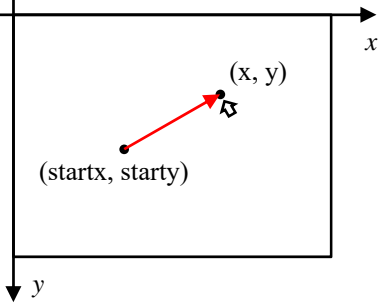
```
void myMouse(int button, int state, int x, int y)  
{  
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {  
        startx = x;  
        starty = y;  
        moving = true;  
    }  
    else {  
        /* 省略 */  
    }  
}
```

左ボタンを押す



```
void myMotion(int x, int y)  
{  
    int diffx, diffy;  
    const double SPEED = 0.5;  
  
    /* 省略 */  
    diffx = x - startx;  
    diffy = y - starty;  
    anglex += (double)diffy * SPEED;  
    angley += (double)diffx * SPEED;  
  
    startx = x;  
    starty = y;  
    glutPostRedisplay();  
}
```

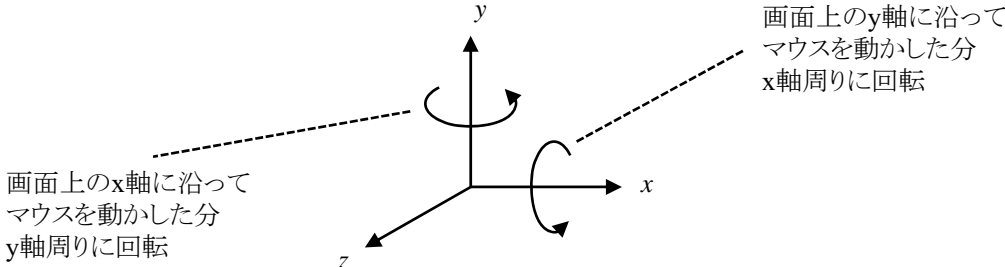
前回の座標(startx, starty)から  
現在の座標(x, y)に移動した  
その移動量を表す



左ボタンを押し下げたまま  
ドラッグする



moving = true



anglexとangleyをもとに座標系を回転させて物体を描画する

```
glRotated(anglex, 1.0, 0.0, 0.0);  
glRotated(angley, 0.0, 1.0, 0.0);
```

moving = false

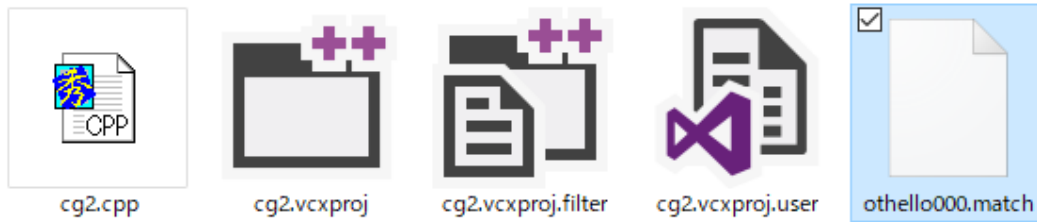
```
void myMouse(int button, int state, int x, int y)  
{  
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {  
        /* 省略 */  
    }  
    else {  
        moving = false;  
    }  
}
```

左ボタンを放す

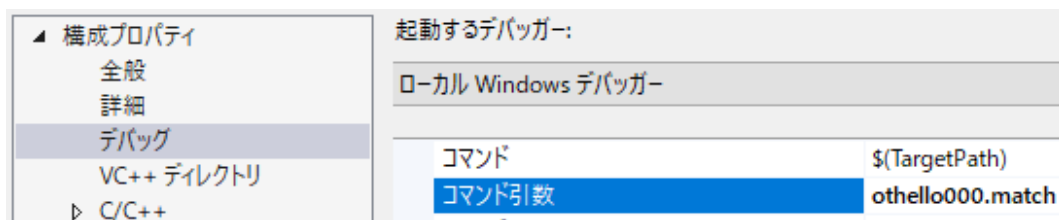


## コマンドライン引数

- ✖ コマンド引数にファイル名を指定して実行しないとイケない
- ✖ まず、配布したデータファイル(以降dice.bmpと呼ぶことにする)をcppファイルのあるディレクトリにコピーする



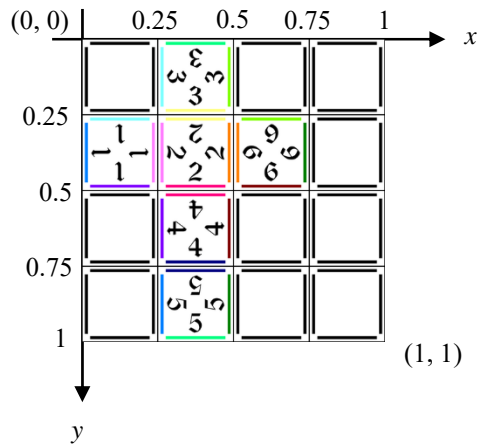
- ✖ Visual Studioでslnファイルを開いて、プロジェクトのプロパティを開く
- ✖ [構成プロパティ]-[デバッグ]-[コマンド引数]でdice.bmpと書く





サイコロへのテクスチャマッピング

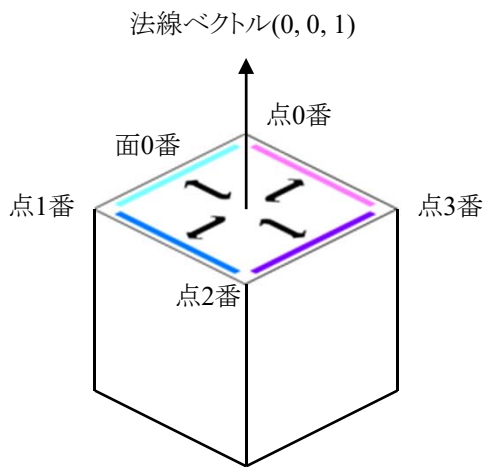
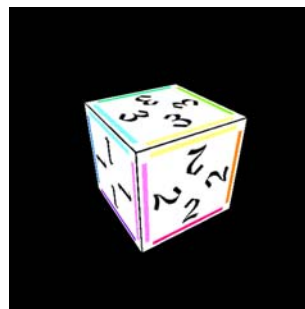
🌸 [課題]サイコロを表示するプログラムを作れ



512×512のbmpファイル

Q. また課題は「座標」に関する事なのか...

A. CGの機能のほとんどはOpenGLが勝手にやってくれるからね。でも、何をどこに配置するかという「座標」に関する事はプログラマがやるしかない。



点ID	テクスチャ座標	頂点座標
0	(0.25, 0.25)	(1, 1, 1)
1	(0, 0.25)	(-1, 1, 1)
2	(0, 0.5)	(-1, -1, 1)
3	(0.25, 0.5)	(1, -1, 1)

座標の一例  
(こうしなければいけないというわけではない。  
自分のやりやすい座標でOK)

## プロトタイプ

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

int winw, winh;
bool moving;
int startx, starty;
double anglex, angley;
int imagew, imageh;
unsigned char* imagedata;

void myCube()
{
    double vlist[8][3] = {
        {1.0, 1.0, 1.0},
        {-1.0, 1.0, 1.0},
        {-1.0, -1.0, 1.0},
        {1.0, -1.0, 1.0},
        {1.0, 1.0, -1.0},
        {-1.0, 1.0, -1.0},
        {-1.0, -1.0, -1.0},
        {1.0, -1.0, -1.0}
    };
    double nlist[6][3] = {
        {0.0, 0.0, 1.0},
        {1.0, 0.0, 0.0},
        {0.0, 1.0, 0.0},
        {0.0, -1.0, 0.0},
        {-1.0, 0.0, 0.0},
        {0.0, 0.0, -1.0}
    };
    int flist[6][4] = {
        {2, 3, 0, 1},
        {3, 7, 4, 0},
        {0, 4, 5, 1},
        {2, 6, 7, 3},
        {1, 5, 6, 2},
        {7, 6, 5, 4}
    };
};
```

## プロトタイプ

```
// [課題] ここから
double tlist[6][4][2] = {
    {
        {0.0, 0.0},
        {0.0, 1.0},
        {1.0, 1.0},
        {1.0, 0.0}
    },
    {
        {0.0, 0.0},
        {0.0, 1.0},
        {1.0, 1.0},
        {1.0, 0.0}
    },
    {
        {0.0, 0.0},
        {0.0, 1.0},
        {1.0, 1.0},
        {1.0, 0.0}
    },
    {
        {0.0, 0.0},
        {0.0, 1.0},
        {1.0, 1.0},
        {1.0, 0.0}
    },
    {
        {0.0, 0.0},
        {0.0, 1.0},
        {1.0, 1.0},
        {1.0, 0.0}
    },
    {
        {0.0, 0.0},
        {0.0, 1.0},
        {1.0, 1.0},
        {1.0, 0.0}
    }
};
// [課題] ここまで
// テクスチャ座標を適切に設定せよ

int i;
int i0, i1, i2, i3;

glBegin(GL_QUADS);

for (i = 0; i < 6; i++) {
    i0 = flist[i][0];
    i1 = flist[i][1];
    i2 = flist[i][2];
    i3 = flist[i][3];
    glNormal3dv(nlist[i]);
    glTexCoord2dv(tlist[i][0]);
    glVertex3dv(vlist[i0]);
    glTexCoord2dv(tlist[i][1]);
    glVertex3dv(vlist[i1]);
    glTexCoord2dv(tlist[i][2]);
    glVertex3dv(vlist[i2]);
    glTexCoord2dv(tlist[i][3]);
    glVertex3dv(vlist[i3]);
}
glEnd();
}
```

## プロトタイプ

```
int readbmp(char* filename, unsigned char** image, int* sizex, int* sizey)
{
    FILE* fp;
    int row;
    unsigned char* ras;
    char header1, header2;
    int imagew, imageh;
    unsigned short bitcount;

    fopen_s(&fp, filename, "rb");
    if (fp == NULL) {
        fprintf(stderr, "ファイル%sを開けません\n", filename);
        return 1;
    }

    fread(&header1, 1, 1, fp);
    fread(&header2, 1, 1, fp);
    if (header1 != 'B' || header2 != 'M') {
        fprintf(stderr, "ファイル%sはBMPファイルではありません\n", filename);
        return 1;
    }
    fseek(fp, 12, 1);

    fseek(fp, 4, 1);
    fread(&imagew, 4, 1, fp);
    fread(&imageh, 4, 1, fp);
    *sizex = imagew;
    *sizey = imageh;

    fseek(fp, 2, 1);
    fread(&bitcount, 2, 1, fp);
    if (bitcount != 24) {
        fprintf(stderr, "ファイル%sは24ビットフルカラーではありません\n", filename);
        return 1;
    }
    fseek(fp, 24, 1);

    row = imagew * 3;
    row = ((row + 3) / 4) * 4;
    ras = (unsigned char*)malloc(sizeof(unsigned char) * row * imageh);
    if (ras == NULL) {
        fprintf(stderr, "メモリを確保できません\n");
        return 1;
    }
    fread(ras, sizeof(unsigned char), row * imageh, fp);

    *image = (unsigned char*)malloc(sizeof(unsigned char) * imagew * imageh * 3);
    if (*image == NULL) {
        fprintf(stderr, "メモリを確保できません\n");
        return 1;
    }
    for (int y = 0; y < imageh; y++) {
        for (int x = 0; x < imagew; x++) {
            int cr, cg, cb;
            int i;
            i = (imageh - 1 - y) * row + x * 3;
            cb = ras[i + 0];
            cg = ras[i + 1];
            cr = ras[i + 2];
            (*image)[y * imagew * 3 + x * 3 + 0] = cr;
            (*image)[y * imagew * 3 + x * 3 + 1] = cg;
            (*image)[y * imagew * 3 + x * 3 + 2] = cb;
        }
    }
    free(ras);
    fclose(fp);
    return 0;
}
```

## プロトタイプ

```
void myDisplay()
{
    unsigned int texName;

    glGenTextures(1, &texName);
    glBindTexture(GL_TEXTURE_2D, texName);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, imagew, imageh, 0, GL_RGB, GL_UNSIGNED_BYTE, imagedata);
    glEnable(GL_TEXTURE_2D);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

    glClearColor(0.0, 0.0, 0.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, (double)winw / (double)winh, 0.1, 100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslated(0.0, 0.0, -5.0);
    glRotated(anglex, 1.0, 0.0, 0.0);
    glRotated(angley, 0.0, 1.0, 0.0);

    myCube();

    glDeleteTextures(1, &texName);
    glDisable(GL_TEXTURE_2D);

    glutSwapBuffers();
}

void myMouse(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        startx = x;
        starty = y;
        moving = true;
    }
    else {
        moving = false;
    }
}

void myMotion(int x, int y)
{
    int diffx, diffy;
    const double SPEED = 0.5;

    if (!moving) return;
    diffx = x - startx;
    diffy = y - starty;
    anglex += (double)diffy * SPEED;
    angley += (double)diffx * SPEED;

    startx = x;
    starty = y;
    glutPostRedisplay();
}

void myReshape(int width, int height)
{
    winw = width;
    winh = height;
    glViewport(0, 0, winw, winh);
}

void myKeyboard(unsigned char key, int x, int y)
{
    if (key == 0x1B) exit(0);
}
```

## プロトタイプ

```
void myInit(char* progname)
{
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow(progname);
}

int main(int argc, char* argv[])
{
    imagedata = NULL;
    anglex = 0.0;
    angley = 0.0;
    moving = false;
    if (argc <= 1) {
        printf("filename not specified\n");
        return 1;
    }
    if (readbmp(argv[1], &imagedata, &imagew, &imageh)) {
        return 1;
    }

    glutInit(&argc, argv);
    myInit(argv[0]);
    glutKeyboardFunc(myKeyboard);
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutMotionFunc(myMotion);
    glutDisplayFunc(myDisplay);
    glutMainLoop();
    return 0;
}
```

## バーチャルトラックボール1

- ✖ 課題にあるバーチャルトラックボールの操作はどこかおかしいです
  - ✖ 実際に使ってみると分かります
- ✖ というわけで、余裕のある人は次ページ以降を実装しましょう

## バーチャルトラックボール2

- ✖ バーチャルトラックボールの別の実装方法
- ✖ 課題にあるバーチャルトラックボールの操作に不満がある人向け
- ✖ 説明は省略

グローバル変数double R[16];を追加

```
main関数の中にRの初期化部分を入れる (ただし, int i;を宣言すること)
for(i = 0; i < 16; i++) R[i] = 0.0;
R[0] = R[5] = R[10] = R[15] = 1.0;
```

glRotated(anglex, 1.0, 0.0, 0.0); glRotated(angley, 0.0, 1.0, 0.0);の代わりにglMultMatrixd(R);に変更

以下のmyRotate関数を追加

```
void myRotate(double angle, double x, double y, double z)
{
    static double Rtmp[3][3];
    static double Rin[3][3];
    double r = sqrt(x * x + y * y + z * z);
    if (r > 0.0) {
        x /= r;
        y /= r;
        z /= r;
    }
    double c = cos(angle / 180.0 * M_PI);
    double s = sin(angle / 180.0 * M_PI);
    Rtmp[0][0] = x * x * (1.0 - c) + c;
    Rtmp[1][1] = y * y * (1.0 - c) + c;
    Rtmp[2][2] = z * z * (1.0 - c) + c;
    Rtmp[0][1] = x * y * (1.0 - c) - s * z;
    Rtmp[1][0] = y * x * (1.0 - c) + s * z;
    Rtmp[0][2] = x * z * (1.0 - c) + s * y;
    Rtmp[2][0] = z * x * (1.0 - c) - s * y;
    Rtmp[1][2] = y * z * (1.0 - c) - s * x;
    Rtmp[2][1] = z * y * (1.0 - c) + s * x;
    Rin[0][0] = R[0];
    Rin[1][0] = R[1];
    Rin[2][0] = R[2];
    Rin[0][1] = R[4];
    Rin[1][1] = R[5];
    Rin[2][1] = R[6];
    Rin[0][2] = R[8];
    Rin[1][2] = R[9];
    Rin[2][2] = R[10];
    R[0] = Rtmp[0][0] * Rin[0][0] + Rtmp[0][1] * Rin[1][0] + Rtmp[0][2] * Rin[2][0];
    R[4] = Rtmp[0][0] * Rin[0][1] + Rtmp[0][1] * Rin[1][1] + Rtmp[0][2] * Rin[2][1];
    R[8] = Rtmp[0][0] * Rin[0][2] + Rtmp[0][1] * Rin[1][2] + Rtmp[0][2] * Rin[2][2];
    R[1] = Rtmp[1][0] * Rin[0][0] + Rtmp[1][1] * Rin[1][0] + Rtmp[1][2] * Rin[2][0];
    R[5] = Rtmp[1][0] * Rin[0][1] + Rtmp[1][1] * Rin[1][1] + Rtmp[1][2] * Rin[2][1];
    R[9] = Rtmp[1][0] * Rin[0][2] + Rtmp[1][1] * Rin[1][2] + Rtmp[1][2] * Rin[2][2];
    R[2] = Rtmp[2][0] * Rin[0][0] + Rtmp[2][1] * Rin[1][0] + Rtmp[2][2] * Rin[2][0];
    R[6] = Rtmp[2][0] * Rin[0][1] + Rtmp[2][1] * Rin[1][1] + Rtmp[2][2] * Rin[2][1];
    R[10] = Rtmp[2][0] * Rin[0][2] + Rtmp[2][1] * Rin[1][2] + Rtmp[2][2] * Rin[2][2];
}
```

myMotion関数の中のanglex += (double)diffy \* SPEED; angley += (double)diffx \* SPEED;の代わりに以下に差し替える  
(ただし, double rotx, roty;を宣言すること)

```
rotx = (double)diffx * SPEED;
roty = (double)-diffy * SPEED;
myRotate(sqrt(rotx * rotx + roty * roty), -roty, rotx, 0.0);
```

- ✖ バージョナルトラックボールの別の実装方法
- ✖ 左ボタンによる回転と右ボタンによる回転の両方で操作する
- ✖ 説明は省略

```

#pragma warning(disable:4996)
#include <stdlib.h>
#include <stdio.h>
#define _USE_MATH_DEFINES
#include <math.h>
#include <GL/glut.h>

int winw, winh;
const int MY_NONE = 0;
const int MY_ROTATE1 = 1;
const int MY_ROTATE2 = 2;
int mousemode;
int startx, starty;
int endx, endy;
int imagew, imageh;
unsigned char* imagedata;
double idleR[16];
double dragR[16];
const double mouseSpeed = 0.5;

void myRotate(double* outR, double* inR, double angle, double x, double y, double z)
{
    static double lR[3][3];
    static double rR[3][3];
    double r = sqrt(x * x + y * y + z * z);
    if (r > 0.0) {
        x /= r;
        y /= r;
        z /= r;
    }
    double c = cos(angle / 180.0 * M_PI);
    double s = sin(angle / 180.0 * M_PI);
    lR[0][0] = x * x * (1.0 - c) + c;
    lR[1][1] = y * y * (1.0 - c) + c;
    lR[2][2] = z * z * (1.0 - c) + c;
    lR[0][1] = x * y * (1.0 - c) - s * z;
    lR[1][0] = y * x * (1.0 - c) + s * z;
    lR[0][2] = x * z * (1.0 - c) + s * y;
    lR[2][0] = z * x * (1.0 - c) - s * y;
    lR[1][2] = y * z * (1.0 - c) - s * x;
    lR[2][1] = z * y * (1.0 - c) + s * x;
    rR[0][0] = inR[0];
    rR[1][0] = inR[1];
    rR[2][0] = inR[2];
    rR[0][1] = inR[4];
    rR[1][1] = inR[5];
    rR[2][1] = inR[6];
    rR[0][2] = inR[8];
    rR[1][2] = inR[9];
    rR[2][2] = inR[10];
    outR[0] = lR[0][0] * rR[0][0] + lR[0][1] * rR[1][0] + lR[0][2] * rR[2][0];
    outR[4] = lR[0][0] * rR[0][1] + lR[0][1] * rR[1][1] + lR[0][2] * rR[2][1];
    outR[8] = lR[0][0] * rR[0][2] + lR[0][1] * rR[1][2] + lR[0][2] * rR[2][2];
    outR[1] = lR[1][0] * rR[0][0] + lR[1][1] * rR[1][0] + lR[1][2] * rR[2][0];
    outR[5] = lR[1][0] * rR[0][1] + lR[1][1] * rR[1][1] + lR[1][2] * rR[2][1];
    outR[9] = lR[1][0] * rR[0][2] + lR[1][1] * rR[1][2] + lR[1][2] * rR[2][2];
    outR[2] = lR[2][0] * rR[0][0] + lR[2][1] * rR[1][0] + lR[2][2] * rR[2][0];
    outR[6] = lR[2][0] * rR[0][1] + lR[2][1] * rR[1][1] + lR[2][2] * rR[2][1];
    outR[10] = lR[2][0] * rR[0][2] + lR[2][1] * rR[1][2] + lR[2][2] * rR[2][2];
}

```



### バーチャルトラックボール3

```
void myCube()
{
    // 省略
}

int readbmp(char* filename, unsigned char** image, int* sizex, int* sizey)
{
    // 省略
}

void myDisplay()
{
    float diffuse[] = { 0.8f, 0.6f, 1.0f, 1.0f };
    unsigned int texName;

    glGenTextures(1, &texName);
    glBindTexture(GL_TEXTURE_2D, texName);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, imagew, imageh, 0, GL_RGB, GL_UNSIGNED_BYTE, imagedata);
    glEnable(GL_TEXTURE_2D);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

    glClearColor(0.0, 0.0, 0.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, (double)winw / (double)winh, 0.1, 100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslated(0.0, 0.0, -5.0);
    if (mousemode == MY_NONE) glMultMatrixd(idler);
    else glMultMatrixd(dragR);

    glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse);
    myCube();

    glDeleteTextures(1, &texName);
    glDisable(GL_TEXTURE_2D);

    glDisable(GL_LIGHT0);
    glDisable(GL_LIGHTING);
    glDisable(GL_DEPTH_TEST);

    glLoadIdentity();
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();

    glColor3d(1.0, 0.0, 0.0);
    glBegin(GL_LINES);
    if (mousemode == MY_ROTATE1) {
        glVertex3d(2.0 * (double)startx / (double)winw - 1.0, -(2.0 * (double)starty / (double)winh - 1.0),
0.0);
        glVertex3d(2.0 * (double)endx / (double)winw - 1.0, -(2.0 * (double)endy / (double)winh - 1.0), 0.0);
    }
    if (mousemode == MY_ROTATE2) {
        glVertex3d(0.0, 0.0, 0.0);
        glVertex3d(2.0 * (double)endx / (double)winw - 1.0, -(2.0 * (double)endy / (double)winh - 1.0), 0.0);
    }
    glEnd();

    glPopMatrix();

    glutSwapBuffers();
}
```

```

void myMouse(int button, int state, int x, int y)
{
    int xdis, ydis;
    double xrot, yrot;
    double vstartx, vstarty, vstartz;
    double r;
    double vendx, vendy, vendz;
    double crossx, crossy, crossz;
    double dot;

    if (state == GLUT_DOWN) {
        startx = x;
        starty = y;
        endx = x;
        endy = y;
        if (button == GLUT_LEFT_BUTTON) {
            mousemode = MY_ROTATE1;
            glutPostRedisplay();
        }
        if (button == GLUT_RIGHT_BUTTON) {
            mousemode = MY_ROTATE2;
            glutPostRedisplay();
        }
    }
    if (state == GLUT_UP) {
        if (mousemode == MY_ROTATE1) {
            xdis = x - startx;
            ydis = y - starty;
            xrot = (double)xdis * mouseSpeed;
            yrot = (double)-ydis * mouseSpeed;
            myRotate(idleR, idleR, sqrt(xrot * xrot + yrot * yrot), -yrot, xrot, 0.0);
            mousemode = MY_NONE;
            glutPostRedisplay();
        }
        if (mousemode == MY_ROTATE2) {
            endx = x;
            endy = y;
            vstartx = (double)(startx - winw / 2);
            vstarty = (double)(starty - winh / 2);
            vstartz = 0.0;
            r = sqrt(vstartx * vstartx + vstarty * vstarty + vstartz * vstartz);
            if (r > 0.0) {
                vstartx /= r;
                vstarty /= r;
                vstartz /= r;
            }
            vendx = (double)(endx - winw / 2);
            vendy = (double)(endy - winh / 2);
            vendz = 0.0;
            r = sqrt(vendx * vendx + vendy * vendy + vendz * vendz);
            if (r > 0.0) {
                vendx /= r;
                vendy /= r;
                vendz /= r;
            }
            crossx = vstarty * vendz - vstartz * vendy;
            crossy = vstartz * vendx - vstartx * vendz;
            crossz = vstartx * vendy - vstarty * vendx;
            dot = vstartx * vendx + vstarty * vendy + vstartz * vendz;
            myRotate(idleR, idleR, atan2(crossz, dot) / M_PI * 180.0, 0.0, 0.0, -1.0);
            mousemode = MY_NONE;
            glutPostRedisplay();
        }
    }
}

```

### バーチャルトラックボール3

```
void myMotion(int x, int y)
{
    int xdis, ydis;
    double xrot, yrot;
    double vstartx, vstarty, vstartz;
    double r;
    double vendx, vendy, vendz;
    double crossx, crossy, crossz;
    double dot;

    endx = x;
    endy = y;
    if (mousemode == MY_ROTATE1) {
        xdis = endx - startx;
        ydis = endy - starty;
        xrot = (float)xdis * mouseSpeed;
        yrot = (float)-ydis * mouseSpeed;
        myRotate(dragR, idleR, sqrt(xrot * xrot + yrot * yrot), -yrot, xrot, 0.0);
        glutPostRedisplay();
    }
    if (mousemode == MY_ROTATE2) {
        vstartx = (double)(startx - winw / 2);
        vstarty = (double)(starty - winh / 2);
        vstartz = 0.0;
        r = sqrt(vstartx * vstartx + vstarty * vstarty + vstartz * vstartz);
        if (r > 0.0) {
            vstartx /= r;
            vstarty /= r;
            vstartz /= r;
        }
        vendx = (double)(endx - winw / 2);
        vendy = (double)(endy - winh / 2);
        vendz = 0.0;
        r = sqrt(vendx * vendx + vendy * vendy + vendz * vendz);
        if (r > 0.0) {
            vendx /= r;
            vendy /= r;
            vendz /= r;
        }
        crossx = vstarty * vendz - vstartz * vendy;
        crossy = vstartz * vendx - vstartx * vendz;
        crossz = vstartx * vendy - vstarty * vendx;
        dot = vstartx * vendx + vstarty * vendy + vstartz * vendz;
        myRotate(dragR, idleR, atan2(crossz, dot) / M_PI * 180.0, 0.0, 0.0, -1.0);
        glutPostRedisplay();
    }
}
```

```
void myReshape(int width, int height)
{
    winw = width;
    winh = height;
    glViewport(0, 0, width, height);
}

void myKeyboard(unsigned char key, int x, int y)
{
    if (key == 0x1B) exit(0);
}

void myInit(char *programe)
{
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize(winw, winh);
    glutInitWindowPosition(0, 0);
    glutCreateWindow(programe);
}

int main(int argc, char* argv[])
{
    int i;
    char line[256];
    imagedata = NULL;
    mousemode = MY_NONE;
    winw = 500;
    winh = 500;
    for (i = 0; i < 16; i++) {
        idleR[i] = 0.0;
        dragR[i] = 0.0;
    }
    idleR[0] = idleR[5] = idleR[10] = idleR[15] = 1.0;
    dragR[0] = dragR[5] = dragR[10] = dragR[15] = 1.0;
    if (argc <= 1) {
        printf("filename not specified\n");
        printf("push any key and push enter\n");
        scanf_s("%255s", line, 256);
        return 1;
    }
    if (readFile(argv[1])) {
        printf("push any key and push enter\n");
        scanf_s("%255s", line, 256);
        return 1;
    }

    glutInit(&argc, argv);
    myInit(argv[0]);
    glutKeyboardFunc(myKeyboard);
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutMotionFunc(myMotion);
    glutDisplayFunc(myDisplay);
    glutMainLoop();
    return 0;
}
```

**注意！**

**PDFファイルではバックスラッシュと円マークの文字コードが違う！**  
**PDFファイルのテキストをそのままコピーするときは**  
**円マークだけはちゃんと自分でキーボードから打つこと！**