OpenGL

- typedef unsigned int GLenum
- typedef unsigned int GLbitfield
- typedef int GLint
- typedef int GLsizei
- typedef unsigned int GLuint
- typedef float GLfloat
- typedef float GLclampf
- typedef double GLdouble
- typedef void GLvoid
- void glutInit(int *argcp, char **argv)
  - GLUTライブラリを初期化します.
  - 「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.
- void glutInitDisplayMode(unsigned int mode)
  - ディスプレイの表示モードを設定します.
  - 「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.
- void glutInitWindowSize(int width, int height)
  - ウィンドウの初期サイズを設定します.
  - 「width」はウィンドウの幅，「height」はウィンドウの高さになります.
- void glutInitWindowPosition(int x, int y)
  - 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.
- int glutCreateWindow(char *title)
  - 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.
- void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)
  - 「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.
  - 「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.
  - 最後の「alpha」は「α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.
- void glutMainLoop(void)
  - GLUTのイベントが発生するまで，待機状態になります.
- void glutSwapBuffers(void)
  - 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.
- void glClear(GLbitfield mask)
  - 「mask」に指定したバッファのビットを初期化します.
  - 「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.
- void glEnable(GLenum cap)
  - GLenum型の引数「cap」に指定した機能を使用可能にします.
  - 「glEnable(GL_DEPTH_TEST)」を実行すると，それ以降「Zバッファ」を使います.
  - 「glEnable(GL_LIGHTING)」を実行すると，それ以降「陰影付け」の計算をします.
  - 「glEnable(GL_LIGHT0)」を実行すると，0番目の光源を点灯します.
  - 「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.

OpenGL

- 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「離した」（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．

OpenGL

- void glMatrixMode(GLenum mode)
  - 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．
- void glLoadIdentity(void)
  - 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．
- 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．
- void glTranslated(GLdouble x, GLdouble y, GLdouble z)
  - 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．
- void glRotated(GLdouble angle, GLdouble x, GLdouble y, GLdouble z)
  - 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．
  - 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．

OpenGL

- void glBegin(GLenum mode)
  void glEnd(void)
  - 「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．
  - 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．
- void glVertex3d(GLdouble x, GLdouble y, GLdouble z)
  - 3次元の座標値を設定します．
  - 引数はGLdouble型の (x, y, z)で指定します．
- void glNormal3d(GLdouble nx, GLdouble ny, GLdouble nz)
  - 単位法線ベクトルを設定します．
  - 引数はGLdouble型の(nx, ny, nz) で指定します．
- void glTexCoord2d(GLdouble s, GLdouble t)
  - 2次元のテクスチャ座標を設定します．
  - 引数はGLdouble型の(s, t) で指定します．
- void glMaterialfv(GLenum face, GLenum pname, GLfloat *params)
  - 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．
  - 「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．

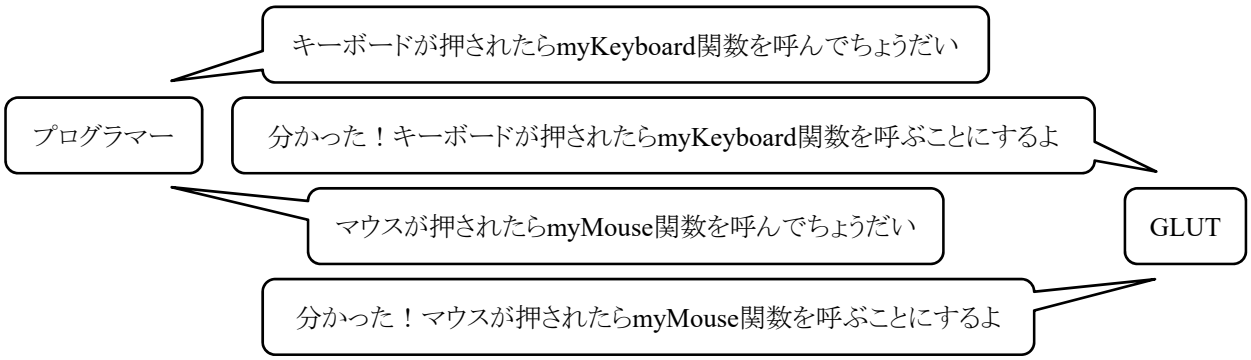
OpenGL

- 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 glTexEnvi(GLenum target, GLenum pname, GLint 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 glTexParameteri(GLenum target, GLenum pname, GLint 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
    - α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
    - α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

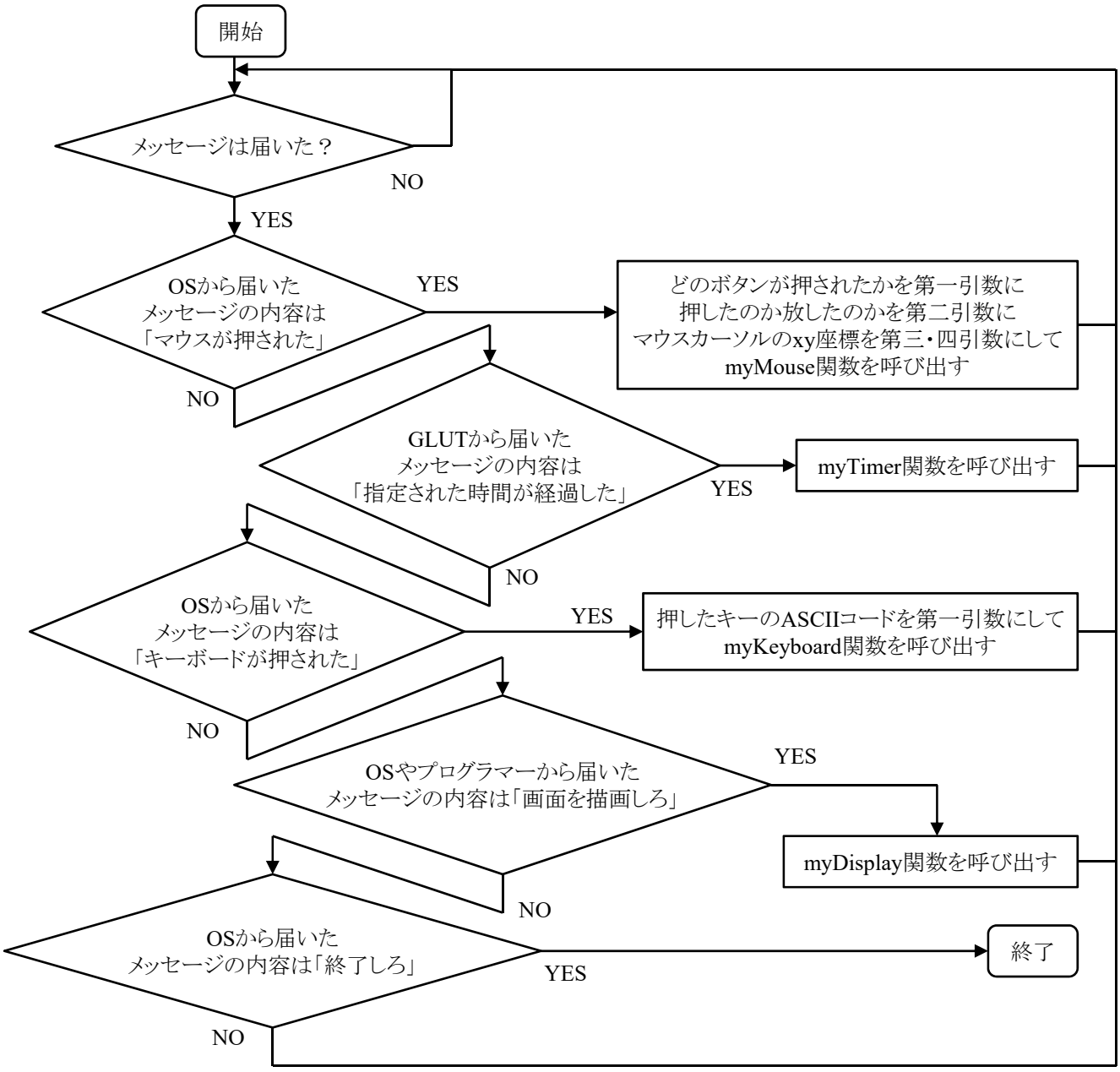
イベントドリブン型プログラミング

- 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



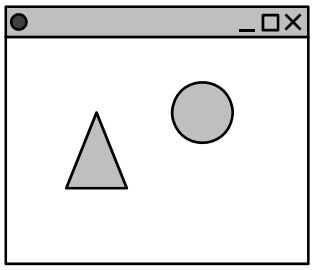
ループ中の動作の例



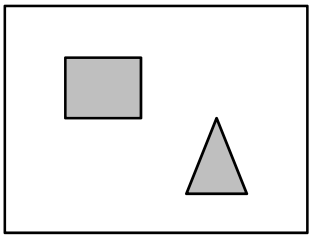
画面の描画

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

描画の流れ

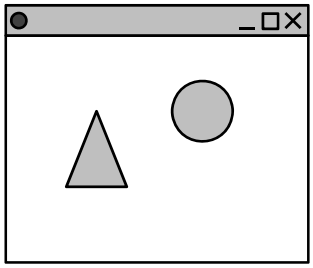


ウィンドウ表示用のバッファ



メモリ内にあるバッファ

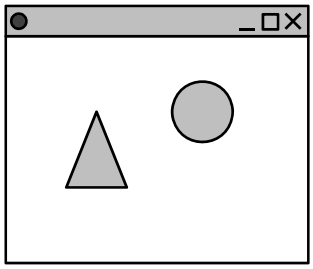
glClear(GL_COLOR_BUFFER_BIT);
画面消去



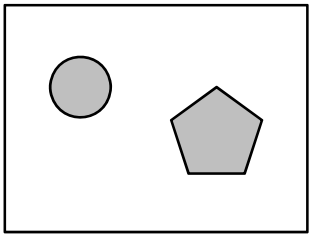
ウィンドウ表示用のバッファ

メモリ内にあるバッファ

glBegin～glEnd
図形の描画

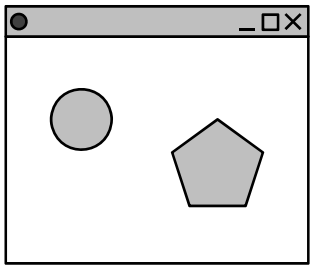


ウィンドウ表示用のバッファ

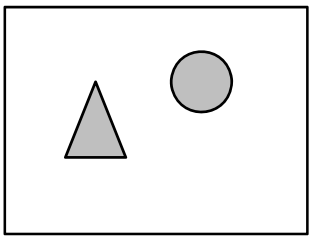


メモリ内にあるバッファ

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

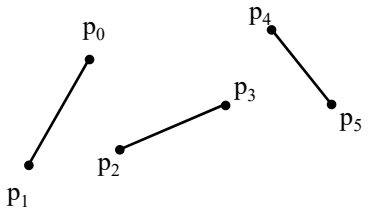
- 点，線，ポリゴンを描くのに，次のようにglBegin()とglEnd()および，glVertex*()を用いる．
- glBegin(mode);
  glVertex*(p0);
  glVertex*(p1);
  .....
  glVertex*(pn);
  glEnd();
- modeは描く図形の種類を指定し，p0，p1，…，pnは座標位置を意味する．

modeの種類

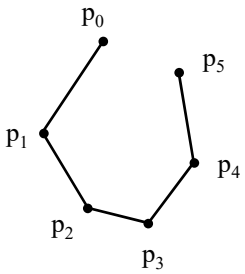
- GL_LINE_STRIP　最初の頂点から最後の頂点まで線分を連結して描画する．
- GL_LINES　二つの頂点を結んだ直線を生成する．
- GL_TRIANGLES　三つ一組の頂点を，それぞれ独立した三角形として描画する．
- GL_QUADS　四つ一組の頂点を，それぞれ独立した四角形として描画する．
- GL_POLYGON　単独の凸ポリゴンを描画する．

ポリゴン描写の注意点

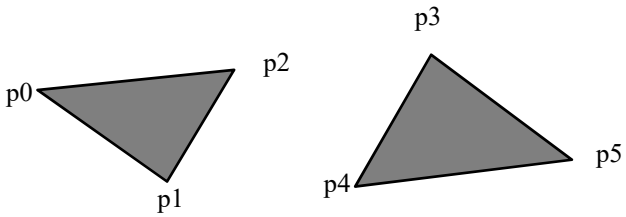
- ポリゴン（polygon）とは多角形の意味
- 頂点座標は反時計回りに設定すること
  - ポリゴンの表面と裏面を区別するため
  - 視点から見て頂点座標が反時計回りに配置されているポリゴンは表面，時計回りの場合は裏面と約束されている



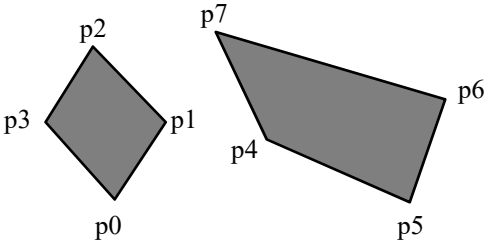
GL_LINES



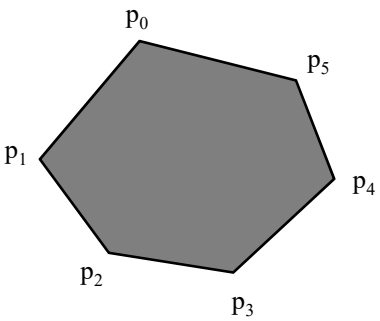
GL_LINE_STRIP



GL_TRIANGLES



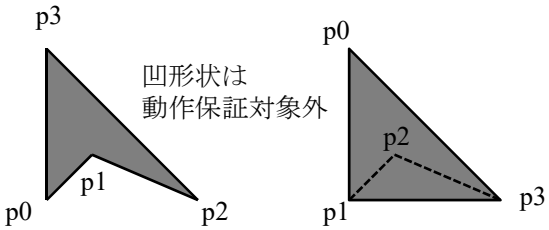
GL_QUADS



GL_POLYGON

四角形の頂点の順番

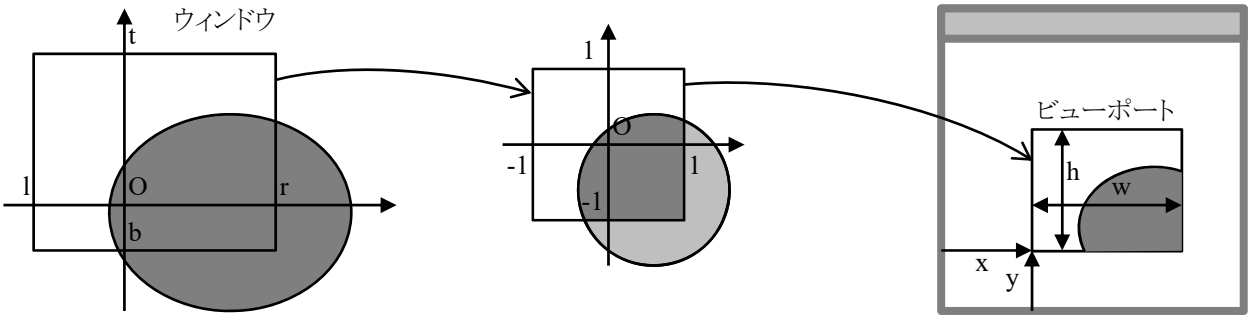
- 4つの頂点の順番によって表示される図形が異なる



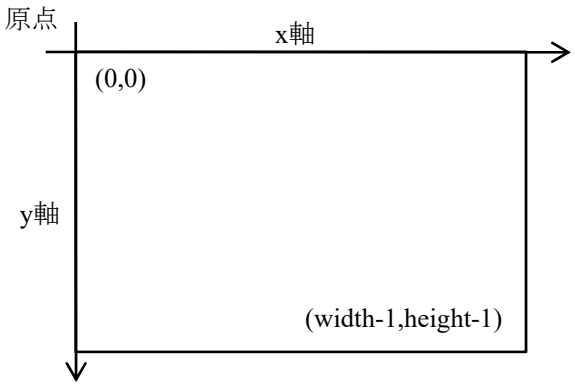
- 右の例は，p0→p1→p2→p3の順番だからおかしくなるのであって，p0→p2→p1→p3の順番でglVertex?d関数を呼び出せば，正しく描画される

- 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



ワールド座標系　　正規化デバイス座標系　　デバイス座標系(ウィンドウ)
(左下が原点)



マウスの座標系

マウス

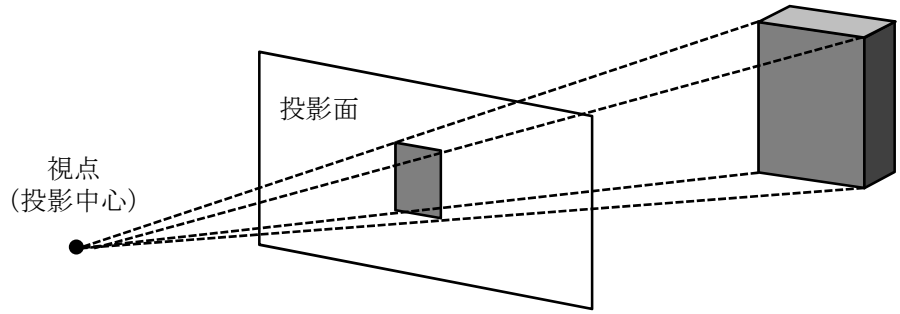
- マウスのボタンを押したか離したか，そのときの画素位置はglutMouseFuncで指定するコールバック関数で確かめる
- マウスをドラッグした画素位置はglutMotionFuncで指定するコールバック関数で確かめる

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



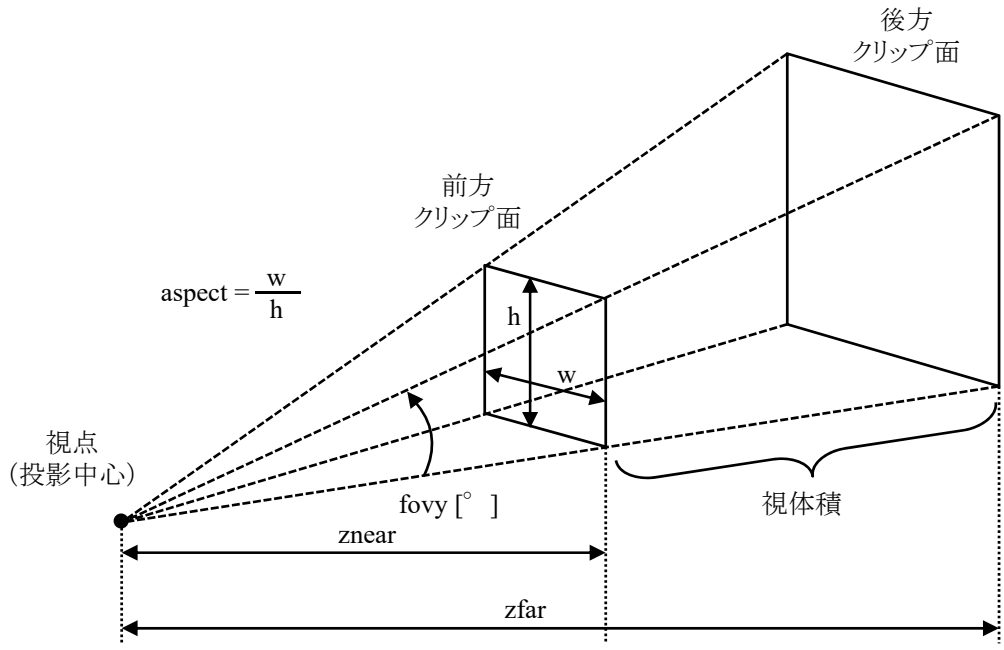
視体積

- 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

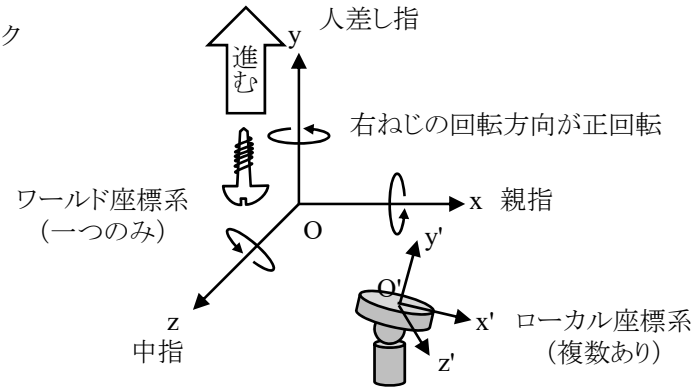
- void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar)
  - 変換行列に透視変換の行列を乗じます．
  - 最初の引数「fovy」はカメラの画角であり，「度」で表します．これが大きいほど広角（透視が強くなり，絵が小さくなります）になり，小さいほど望遠レンズになります．
  - 2つ目の引数「aspect」は画面のアスペクト比（縦横比）であり，「1」であればビューポートに表示される図形のx方向とy方向のスケールが等しくなります．
  - 3つ目の引数「zNear」と4つ目の引数「zFar」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

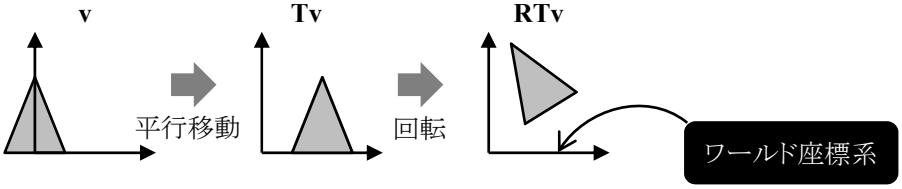
- ワールド座標系：ユーザが描く図形やオブジェクト全てに対して共通な座標系である
- ローカル座標系：各オブジェクト固有の座標系であり，ワールド座標系の中に複数存在する



モデリング変換

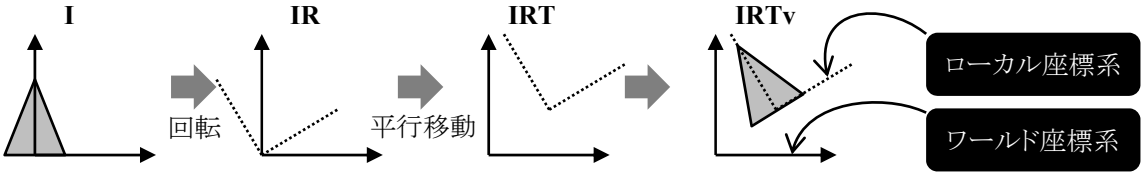
- OpenGLは，オブジェクトの平行移動(translation)，回転(rotation)，スケーリング(scaling)の三つのモデリング変換を用意している
- OpenGLは，2次元や3次元の頂点を扱うが，内部では統一的に4次元ベクトル(x, y, z, w)を用いる．これを同次座標という．
- OpenGLでは，点$\mathbf{v}=(x\ y\ z\ 1)^{\mathrm{T}}$を点$\mathbf{v}'=(x'\ y'\ z'\ 1)^{\mathrm{T}}$に変換する一般式は$\mathbf{v}'=\mathbf{T}\mathbf{v}$で表される
- $\mathbf{T}$は4×4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();
```

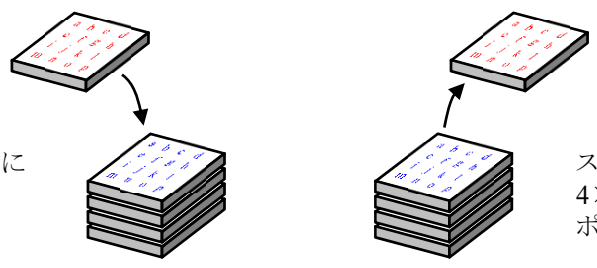
ローカル座標系の考え方



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

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();
```

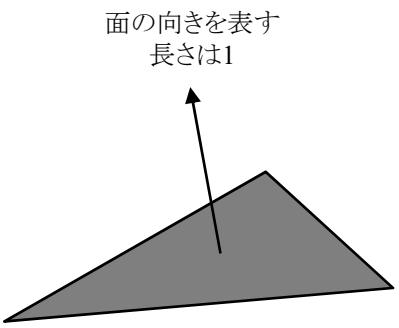
例3
```
glPushMatrix();
glTranslated(posx, posy, 0.1);
glBegin(GL_QUADS);
glVertex3d(-0.5, -0.5, 0.0);
glVertex3d(+0.5, -0.5, 0.0);
glVertex3d(+0.5, +0.5, 0.0);
glVertex3d(-0.5, +0.5, 0.0);
glEnd();
glPopMatrix();
```

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();
```

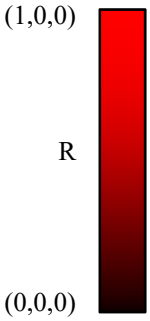


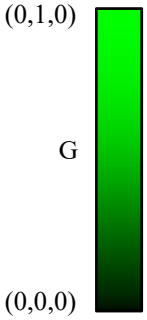
関数の説明

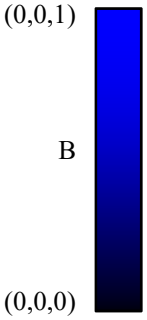
- void glNormal3d(GLdouble nx, GLdouble ny, GLdouble nz);
  - 法線ベクトルを設定します．

色の表現

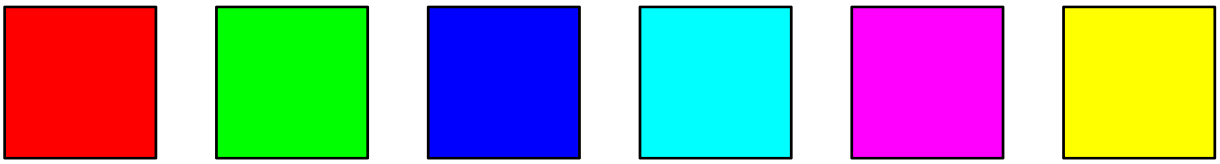
- 光の三原色で表す
- R: Red 赤, G: Green 緑, B: Blue 青
- glClearColor関数で背景色を指定，glColor3d関数で描画する図形の色を指定します
- OpenGLのこれらの関数の引数は，浮動小数点の0～1の値で表します
  - ちなみに，画面や画像ファイルなどの画素は通常，RGBそれぞれ8bitずつ，合計24bitで表します．整数で0～255の値で表します．OpenGLは0～1ですのでご注意ください．
- 0が暗くて，1が明るいです







(R,G,B)=(0,0,0)　　(R,G,B)=(0.5,0.5,0.5)　　(R,G,B)=(1,1,1)



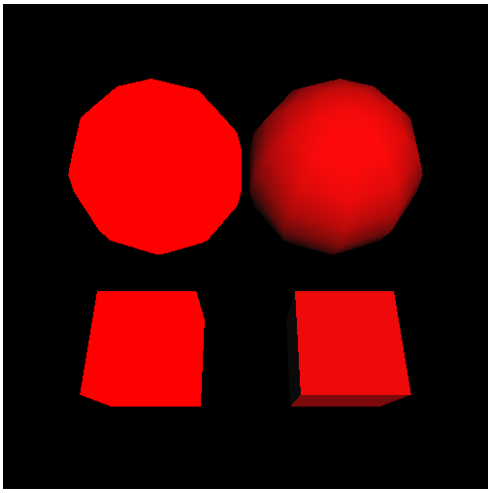
(1,0,0)　(0,1,0)　(0,0,1)　(0,1,1)　(1,0,1)　(1,1,0)

(1,0,0)　(1,0.5,0.5)　(1,1,1)　　(0,1,0)　(0.5,1,0.5)　(1,1,1)　　(0,0,1)　(0.5,0.5,1)　(1,1,1)

(1,0,0)　(1,1,0)　(0,1,0)　(0,1,1)　(0,0,1)　(1,0,1)　(1,0,0)

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

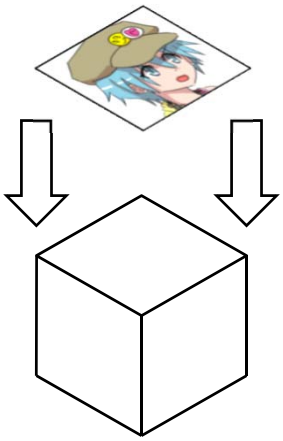
- glEnable(GL_LIGHTING);
  - 光源による陰影を計算する機能をオンにする
- glEnable(GL_LIGHTn);
  - 番号n=0～7の光源を点灯する
- glほにゃらら(引数);
  - 光源が点灯された状態で描画したいシーンのOpenGL命令を実行する
- glDisable(GL_LIGHTn);
  - 番号n=0～7の光源を消灯する
- glDisable(GL_LIGHTING);
  - 光源による陰影を計算する機能をオフにする

物体表面の材質の設定

- void glMaterialfv(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

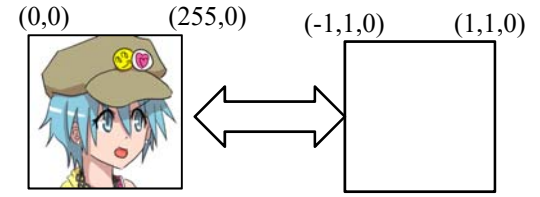
pname	params	パラメータ
GL_DIFFUSE	float型の4次元配列	材質の拡散RGBA値

テクスチャマッピングの座標系

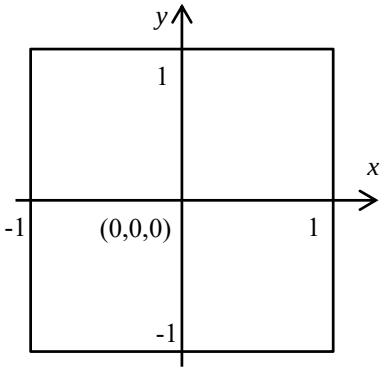


3次元の面に2次元の画像を貼り付ける＝テクスチャマッピング

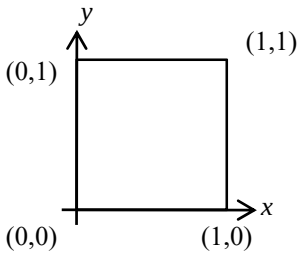
点の座標を対応づける



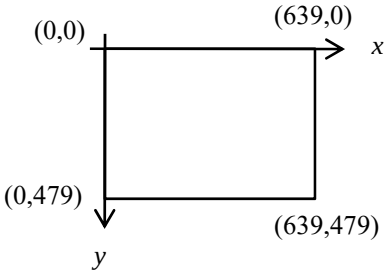
2次元座標　　3次元座標



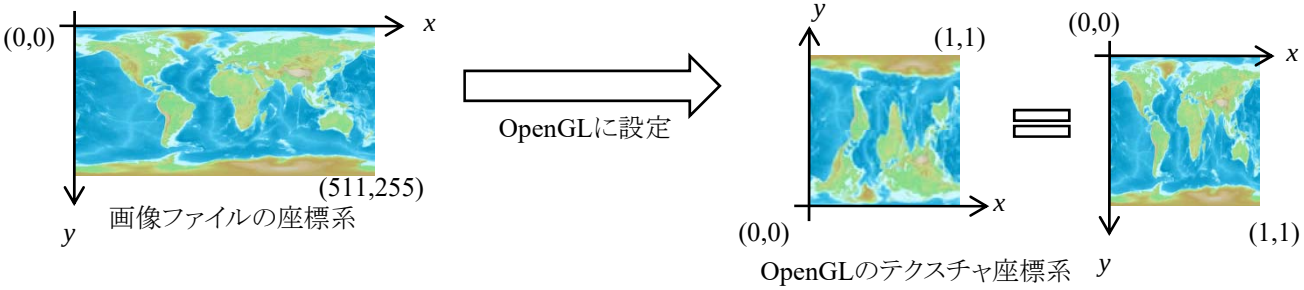
OpenGLの空間の座標系（初期状態）



OpenGLのテクスチャ座標系



画像やディスプレイの座標系



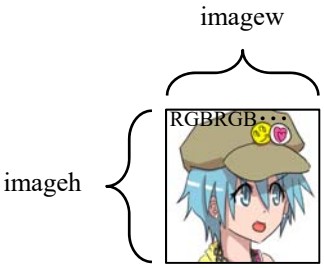

```
glBegin(GL_QUADS);
glTexCoord2d(0.0, 0.0); glVertex3d(-1.0, 1.0, 0.0);
glTexCoord2d(0.0, 1.0); glVertex3d(-1.0, -1.0, 0.0);
glTexCoord2d(1.0, 1.0); glVertex3d(1.0, -1.0, 0.0);
glTexCoord2d(1.0, 0.0); glVertex3d(1.0, 1.0, 0.0);
glEnd();
```

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);
glTexEnvi(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



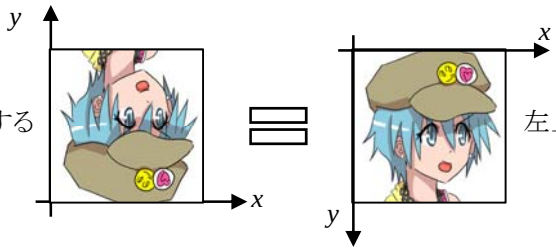
unsigned char
RGBRGB…の順

imagew * imageh * 3バイト

画像の幅と高さは64×64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

左下が原点で上下反転する



左上が原点だと思うと分かりやすい

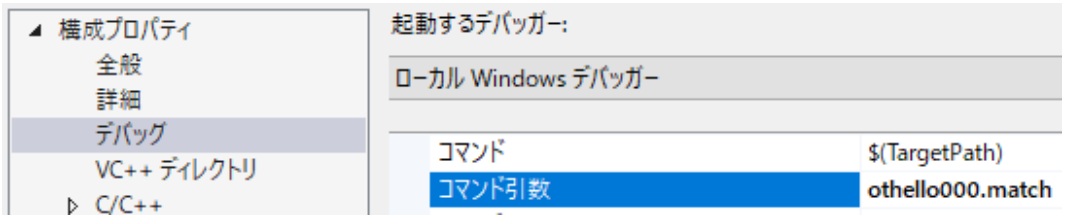
OpenGLの座標系は左下が原点なのだからimagedataは本当は左下から右上に並べるべきだが
このように，左上から右下に並べたほうが分かりやすい

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

コマンドライン引数

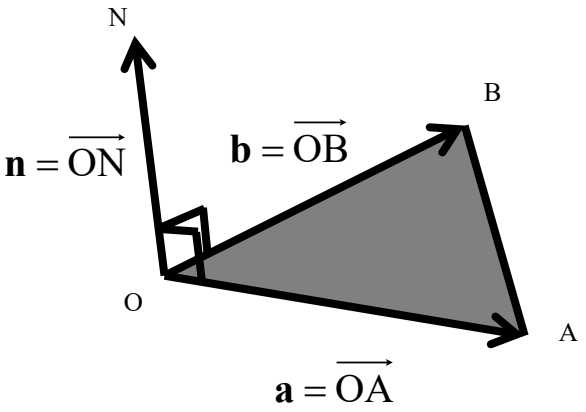
- コマンド引数にファイル名を指定して実行しないといけない
- まず，配布したデータファイル（以降miku.objと呼ぶことにする）をcppファイルのあるディレクトリにコピーする

- 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miku.objと書く



外積

- 3次元ベクトル2つの外積はそれらに直交する(直角に交わる)ベクトルを計算できる. 計算されたベクトルの向きは, 1番目のベクトルから2番目のベクトルに向かう回転を右ネジとみなす方向
- 三角形の法線の向きを計算するのに使う(多角形も複数の三角形で出来ているので計算可能)



三角形OABの法線をNとする
OからA,B,Nへ向かうベクトルをa,b,nと表現する

点の座標を以下のように表現する

$$\mathrm{O}=\begin{pmatrix}\mathrm{O_x}\\ \mathrm{O_y}\\ \mathrm{O_z}\end{pmatrix}\quad \mathrm{A}=\begin{pmatrix}\mathrm{A_x}\\ \mathrm{A_y}\\ \mathrm{A_z}\end{pmatrix}\quad \mathrm{B}=\begin{pmatrix}\mathrm{B_x}\\ \mathrm{B_y}\\ \mathrm{B_z}\end{pmatrix}$$

よって, ベクトルaとbは次のように計算できる
また, 簡単のため, 小文字のax,ay,az,bx,by,bzで表現する

$$\mathbf{a}=\begin{pmatrix}\mathrm{A_x}-\mathrm{O_x}\\ \mathrm{A_y}-\mathrm{O_y}\\ \mathrm{A_z}-\mathrm{O_z}\end{pmatrix}\equiv\begin{pmatrix}a_x\\ a_y\\ a_z\end{pmatrix}\qquad \mathbf{b}=\begin{pmatrix}\mathrm{B_x}-\mathrm{O_x}\\ \mathrm{B_y}-\mathrm{O_y}\\ \mathrm{B_z}-\mathrm{O_z}\end{pmatrix}\equiv\begin{pmatrix}b_x\\ b_y\\ b_z\end{pmatrix}$$

法線Nを計算したい. これはaとbの外積を計算して, 正規化すればいい.
向きに注意すること. 点Oと点Aと点Bが半時計回りになる見え方が表である.
つまり, 外積を計算するときは, a×bを計算すればいい. b×aでは裏側のベクトルになってしまうので注意.
a×bの順番で計算したベクトルをcとおく. cを中心として右ねじの方向に従ってaからbの向きに回転する.
cはaと直交する. cはbと直交する. つまり, cは三角形OABと直交する. つまり, cはこの三角形の法線の向きと同じ.
単位法線ベクトルは長さが1じゃないといけない. cを正規化(長さを1にすること)したのが法線nになる.

外積の公式は以下の通り

$$\mathbf{c}=\begin{pmatrix}c_x\\ c_y\\ c_z\end{pmatrix}=\mathbf{a}\times\mathbf{b}=\begin{pmatrix}a_yb_z-a_zb_y\\ a_zb_x-a_xb_z\\ a_xb_y-a_yb_x\end{pmatrix}$$

ベクトルcの長さをrとおく $r=\|\mathbf{c}\|=\sqrt{c_x^2+c_y^2+c_z^2}$

ベクトルcの長さを1にしたベクトルnを計算する方法は以下の通り

if (r < すごく小さい値) ⇨ エラー！計算できません

else ⇨ $\mathbf{n}=\begin{pmatrix}n_x\\ n_y\\ n_z\end{pmatrix}=\begin{pmatrix}c_x/r\\ c_y/r\\ c_z/r\end{pmatrix}$

データファイルの形式

- データファイルの形式は色々な種類がある(VRML, OBJ, DXF, など)
- 今回はOBJファイル形式を紹介する
- http://www.hiramine.com/programming/3dmodelfileformat/objfileformat.html
- http://wikiwiki.jp/gan/?%A5%E2%A5%C7%A5%EB%A5%C7%A1%BC%A5%BF
- アスキー形式

［ファイルの例］

```
# コメント
mtllib マテリアルファイル名
g グループ名
usemtl マテリアル名
v x成分値 y成分値 z成分値
v x成分値 y成分値 z成分値
v x成分値 y成分値 z成分値
…（省略）…
vt x成分値 y成分値
vt x成分値 y成分値
vt x成分値 y成分値
…（省略）…
vn x成分値 y成分値 z成分値
vn x成分値 y成分値 z成分値
vn x成分値 y成分値 z成分値
…（省略）…
f 頂点座標値番号/テクスチャ座標値番号/頂点法線ベクトル番号（多角形の頂点の数だけ続く）
f 頂点座標値番号/テクスチャ座標値番号/頂点法線ベクトル番号（多角形の頂点の数だけ続く）
f 頂点座標値番号/テクスチャ座標値番号/頂点法線ベクトル番号（多角形の頂点の数だけ続く）
…（省略）…
g グループ名
usemtl マテリアル名
v x成分値 y成分値 z成分値
v x成分値 y成分値 z成分値
v x成分値 y成分値 z成分値
…（省略）…
vt x成分値 y成分値
vt x成分値 y成分値
vt x成分値 y成分値
…（省略）…
vn x成分値 y成分値 z成分値
vn x成分値 y成分値 z成分値
vn x成分値 y成分値 z成分値
…（省略）…
f 頂点座標値番号/テクスチャ座標値番号/頂点法線ベクトル番号（多角形の頂点の数だけ続く）
f 頂点座標値番号/テクスチャ座標値番号/頂点法線ベクトル番号（多角形の頂点の数だけ続く）
f 頂点座標値番号/テクスチャ座標値番号/頂点法線ベクトル番号（多角形の頂点の数だけ続く）
…（省略）…
```

データファイルの形式

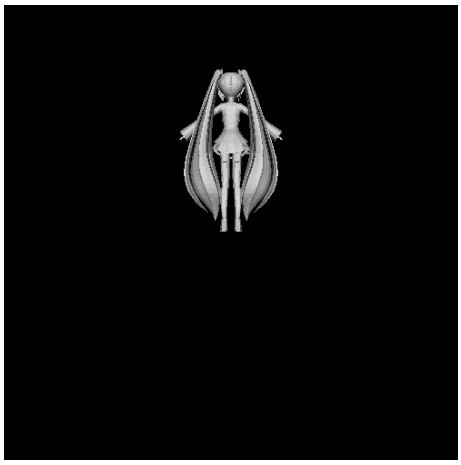
［実際のファイルの例］

```
#                                   ┐
mtllib miku.mtl                     ┘ 無視する
v 1.2 17.6 -0.3                     ┐
v 1.3 17.9 -0.3                     │
v 1.2 17.9 -0.3                     ┘ 頂点の3次元座標
v 1.1 17.6 -0.3
v 1.3 17.9 -0.2
vt 0 1                              ┐
vt 0 1                              │
vt 0 1                              │
vt 0 1                              │
vt 0 1                              │
vn 0.8 -0.3 -0.5                    ┘ 無視する
vn 0.6 -0.1 -0.8
vn 0.5 -0.0 -0.9
vn 0.2 0.0 -1.0
vn 0.5 -0.3 0.8
g obj1
usemtl m0
f 2793/2793/2793 2794/2794/2794 2795/2795/2795   ┐
f 2793/2793/2793 2795/2795/2795 2796/2796/2796   │
f 2796/2796/2796 2797/2797/2797 2798/2798/2798   ┘ 三角形
f 2796/2796/2796 2798/2798/2798 2793/2793/2793
f 2799/2799/2799 2800/2800/2800 2798/2798/2798
usemtl m1
f 1/1/1 2/2/2 3/3/3
f 1/1/1 4/4/4 3/3/3
f 1/1/1 3/3/3 5/5/5
f 1/1/1 4/4/4 3/3/3
f 1/1/1 6/6/6 5/5/5
```

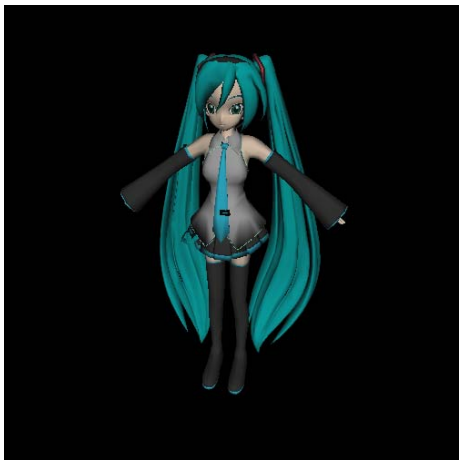
- v 数値x 数値y 数値z
  となっていた場合，頂点の3次元座標(x, y, z)を表す
- f 数値1/数値2/数値3 数値4/数値5/数値6 数値7/数値8/数値9
  となっていた場合，数値1と数値4と数値7の頂点番号によって構成される三角形を表す
- 頂点番号はvの行を上から1, 2, ...と数えたものである
  - 番号は0からではなく1から始まることに注意
- 今回の課題では，三角形のみを読み込み，四角形以上には対応しなくてよい
- 先頭がvの行と先頭がfの行だけを使い，それ以外の行は無視してよい
  - 先頭がvtの行やvnの行も無視することに注意すること

課題

- OBJファイルを読み込むプログラムを作れ



この課題では灰色で後ろ向きに表示されます



色とテクスチャを読み込んで
表示するのはあきらめてください．
とてつもなくややこしいです．
このファイル形式に対応するプログラムを作るだけでも
ややこしいのに，OpenGL特有の問題点もあって，
余計にカオスなことになります．

ファイル処理・文字列処理

- ファイル処理，文字列処理については「C言語の復習」の資料を参照

変数名

- 頂点のデータは変数vlistに入れる
  - vlist[v][0]はv番目（0から始まる値）の頂点のx座標
  - vlist[v][1]はv番目（0から始まる値）の頂点のy座標
  - vlist[v][2]はv番目（0から始まる値）の頂点のz座標
  - 読み込んだ頂点の数はvnum
  - 最大でVMAXまでしか読み込まないようにすること
- 三角形のデータは変数flistに入れる
  - flist[f][0]はf番目（0から始まる値）の三角形の1つ目の頂点の頂点番号（0から始まる値）
  - flist[f][1]はf番目（0から始まる値）の三角形の2つ目の頂点の頂点番号（0から始まる値）
  - flist[f][2]はf番目（0から始まる値）の三角形の3つ目の頂点の頂点番号（0から始まる値）
  - 読み込んだ面の数はfnum
  - 最大でFMAXまでしか読み込まないようにすること

プログラムの流れの一例

```
int readFile(char filename[])
{
    変数の宣言

    filenameというファイルをテキスト読み込みモードで開く
    正しく開くことができなかったらreturn 1

    vnum = fnum = 0;
    while文でファイルの最後の行まで繰り返す {
        if(その１行が"v 浮動小数点x 浮動小数点y 浮動小数点z"のような形式ならば) {
            if(vnum == VMAX) {ファイルを閉じてreturn 1}
            vlist[vnum][0] = 浮動小数点x;
            vlist[vnum][1] = 浮動小数点y;
            vlist[vnum][2] = 浮動小数点z;
            vnum++;
        }
        if(その１行が"f 整数i 整数j 整数k"のような形式ならば) {
            if(fnum == FMAX) {ファイルを閉じてreturn 1}
            flist[fnum][0] = 整数i - 1;
            flist[fnum][1] = 整数j - 1;
            flist[fnum][2] = 整数k - 1;
            fnum++;
    }

    ファイルを閉じてreturn 0
}
```

プロトタイプ

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

int winw, winh;
int vnum, fnum;
#define VMAX 65536
#define FMAX 65536
float vlist[VMAX][3];
int flist[FMAX][3];

int readFile(char filename[])
{
    // [課題] ここから
    FILE* fp;

    fp = fopen(filename, "rt");
    if (fp == NULL) return 1;

    vnum = fnum = 0;

    fclose(fp);
    return 0;
    // [課題] ここまでを適切に修正
}
```

プロトタイプ

```
void myDisplay()
{
    int n;
    int i1, i2, i3;
    double p1x, p1y, p1z;
    double p2x, p2y, p2z;
    double p3x, p3y, p3z;
    double ax, ay, az;
    double bx, by, bz;
    double nx, ny, nz;
    double r;

    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, 0.01, 1000.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslated(0.0, 0.0, -50.0);

    for (n = 0; n < fnum; n++) {
        i1 = flist[n][0];
        i2 = flist[n][1];
        i3 = flist[n][2];
        p1x = (double)vlist[i1][0];
        p1y = (double)vlist[i1][1];
        p1z = (double)vlist[i1][2];
        p2x = (double)vlist[i2][0];
        p2y = (double)vlist[i2][1];
        p2z = (double)vlist[i2][2];
        p3x = (double)vlist[i3][0];
        p3y = (double)vlist[i3][1];
        p3z = (double)vlist[i3][2];
        ax = p2x - p1x;
        ay = p2y - p1y;
        az = p2z - p1z;
        bx = p3x - p1x;
        by = p3y - p1y;
        bz = p3z - p1z;
        nx = ay * bz - az * by;
        ny = az * bx - ax * bz;
        nz = ax * by - ay * bx;
        r = sqrt(nx * nx + ny * ny + nz * nz);
        nx /= r;
        ny /= r;
        nz /= r;

        glBegin(GL_TRIANGLES);
        glNormal3d(nx, ny, nz);
        glVertex3d(p1x, p1y, p1z);
        glVertex3d(p2x, p2y, p2z);
        glVertex3d(p3x, p3y, p3z);
        glEnd();
    }

    glutSwapBuffers();
}

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

プロトタイプ

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

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[])
{
    char line[256];
    if (argc <= 1) {
        printf("filename not specified¥n");
        printf("push any key and push enter¥n");
        scanf("%s", line);
        return 1;
    }
    if (readFile(argv[1])) {
        printf("read error: file '%s'¥n", argv[1]);
        printf("push any key and push enter¥n");
        scanf("%s", line);
        return 1;
    }

    glutInit(&argc, argv);
    myInit(argv[0]);
    glutKeyboardFunc(myKeyboard);
    glutReshapeFunc(myReshape);
    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ペするときは
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！