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.
- void glDisable(GLenum cap)
  - 引数「cap」に指定した機能を使用不可にします.

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．

- 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 glMultMatrixd(GLdouble *m)
  - 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．
  - 16個の値は以下のような4×4行列に対応しています．

$$\begin{pmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{pmatrix}$$

- void glPushMatrix(void)
  - glMatrixMode()で指定している現在の変換行列を保存します．
- void glPopMatrix(void)
  - 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．
- 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) v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)，または，2つの要素を持つGLdouble型の配列v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．

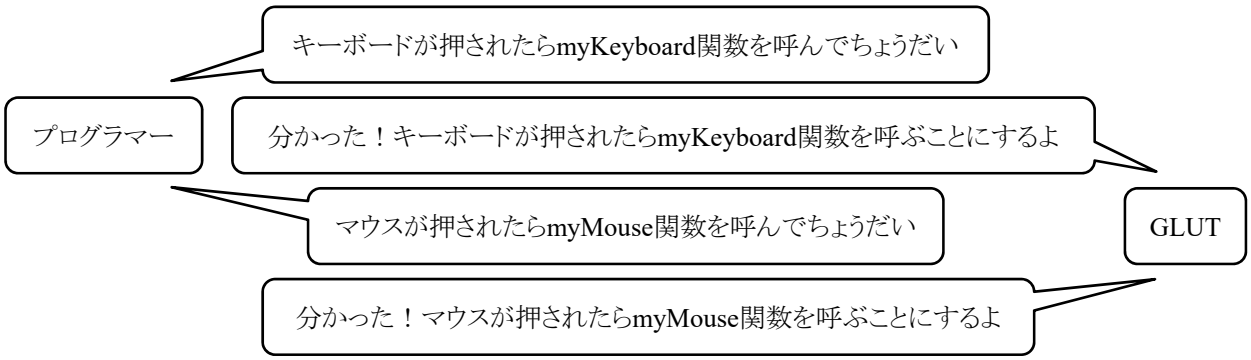
OpenGL

- 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

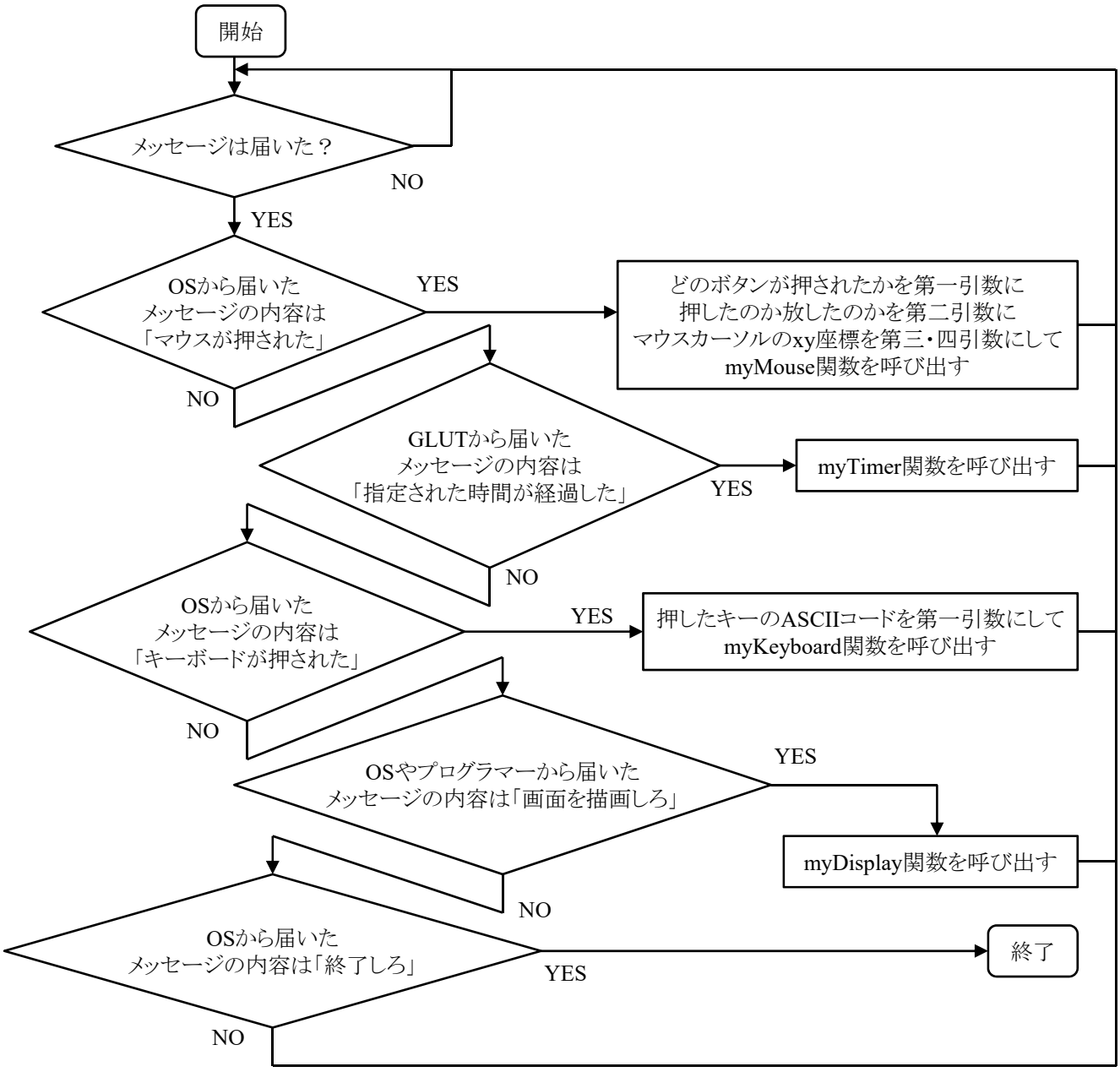
イベントドリブン型プログラミング

- 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



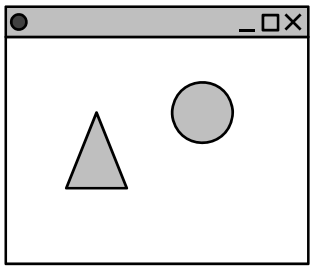
ループ中の動作の例



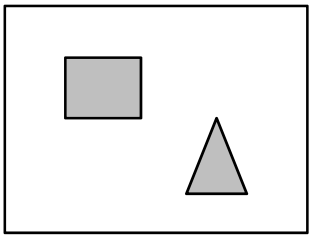
画面の描画

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

描画の流れ

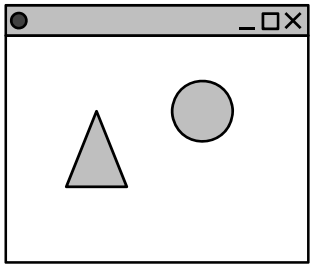


ウィンドウ表示用のバッファ



メモリ内にあるバッファ

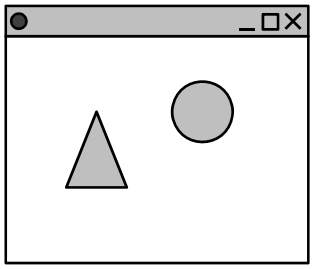
glClear(GL_COLOR_BUFFER_BIT);
画面消去



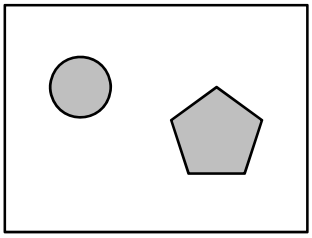
ウィンドウ表示用のバッファ

メモリ内にあるバッファ

glBegin～glEnd
図形の描画

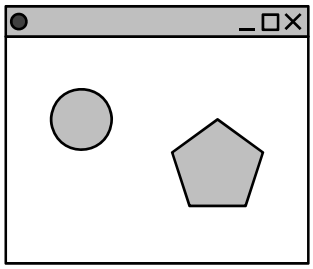


ウィンドウ表示用のバッファ

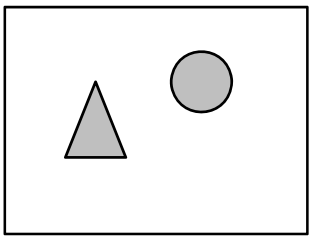


メモリ内にあるバッファ

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

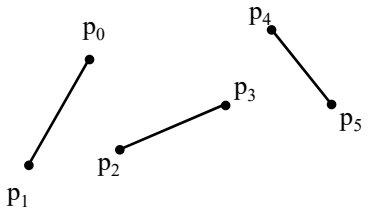
- 点，線，ポリゴンを描くのに，次のようにglBegin()とglEnd()および，glVertex*()を用いる．
- glBegin(mode);
  glVertex*(p0);
  glVertex*(p1);
  .....
  glVertex*(pn);
  glEnd();
- modeは描く図形の種類を指定し，p0，p1，…，pnは座標位置を意味する．

modeの種類

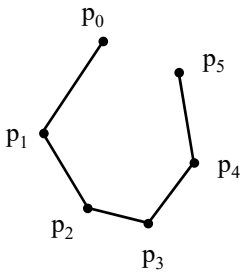
- GL_LINE_STRIP　最初の頂点から最後の頂点まで線分を連結して描画する．
- GL_LINES　二つの頂点を結んだ直線を生成する．
- GL_TRIANGLES　三つ一組の頂点を，それぞれ独立した三角形として描画する．
- GL_QUADS　四つ一組の頂点を，それぞれ独立した四角形として描画する．
- GL_POLYGON　単独の凸ポリゴンを描画する．

ポリゴン描写の注意点

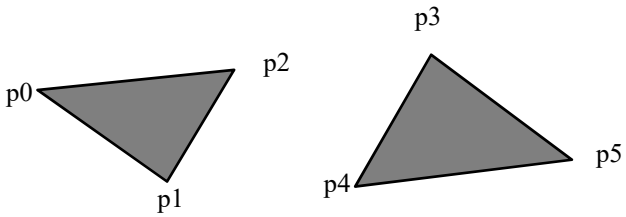
- ポリゴン（polygon）とは多角形の意味
- 頂点座標は反時計回りに設定すること
  - ポリゴンの表面と裏面を区別するため
  - 視点から見て頂点座標が反時計回りに配置されているポリゴンは表面，時計回りの場合は裏面と約束されている



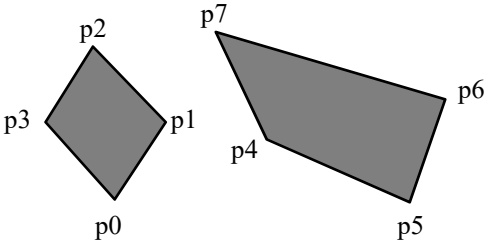
GL_LINES



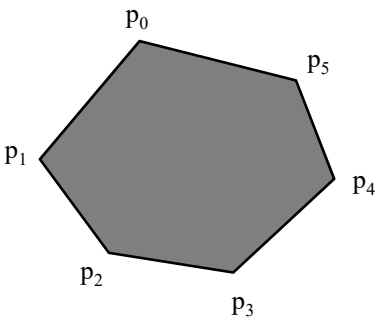
GL_LINE_STRIP



GL_TRIANGLES



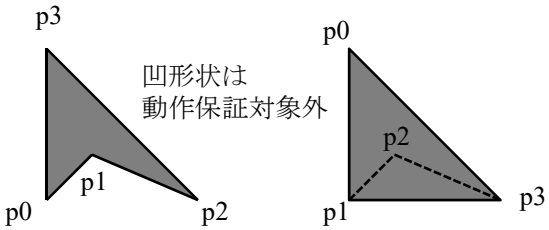
GL_QUADS



GL_POLYGON

四角形の頂点の順番

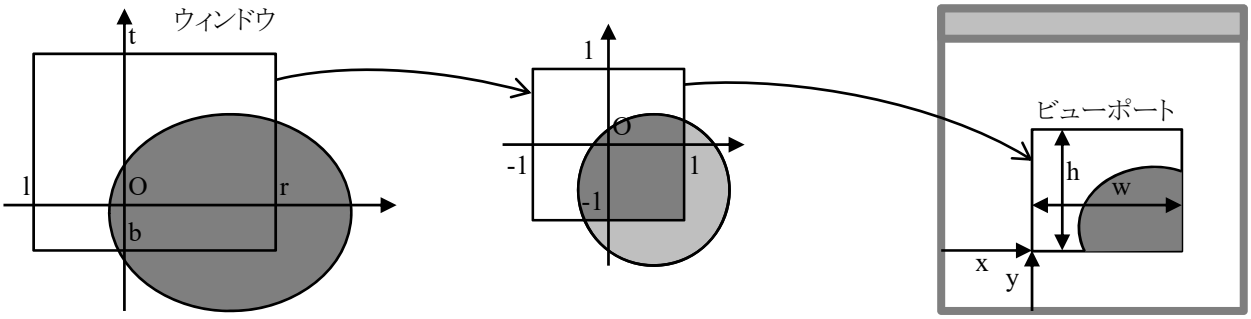
- 4つの頂点の順番によって表示される図形が異なる



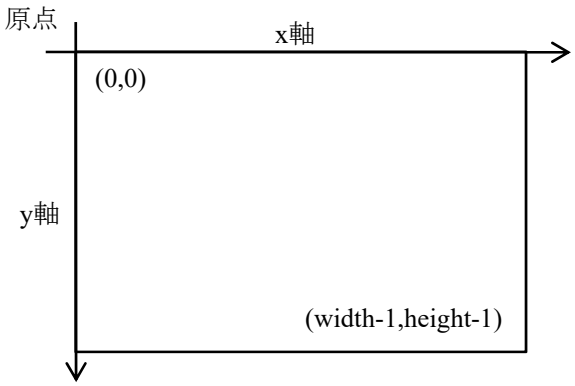
- 右の例は，p0→p1→p2→p3の順番だからおかしくなるのであって，p0→p2→p1→p3の順番でglVertex?d関数を呼び出せば，正しく描画される

- 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

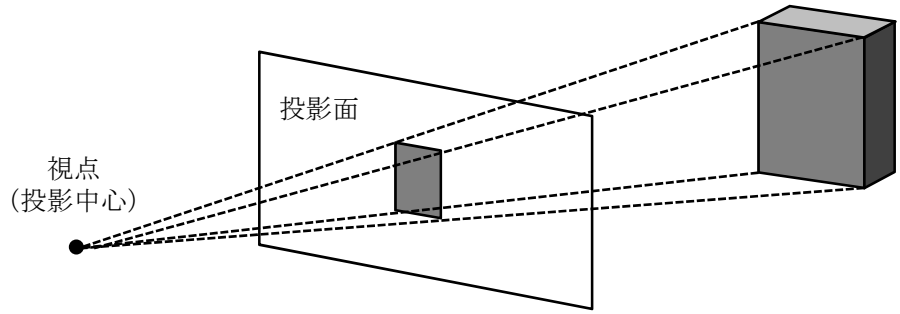
- マウスのボタンを押したか離したか，そのときの画素位置はglutMouseFuncで指定するコールバック関数で確かめる
- マウスをドラッグした画素位置はglutMotionFuncで指定するコールバック関数で確かめる

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



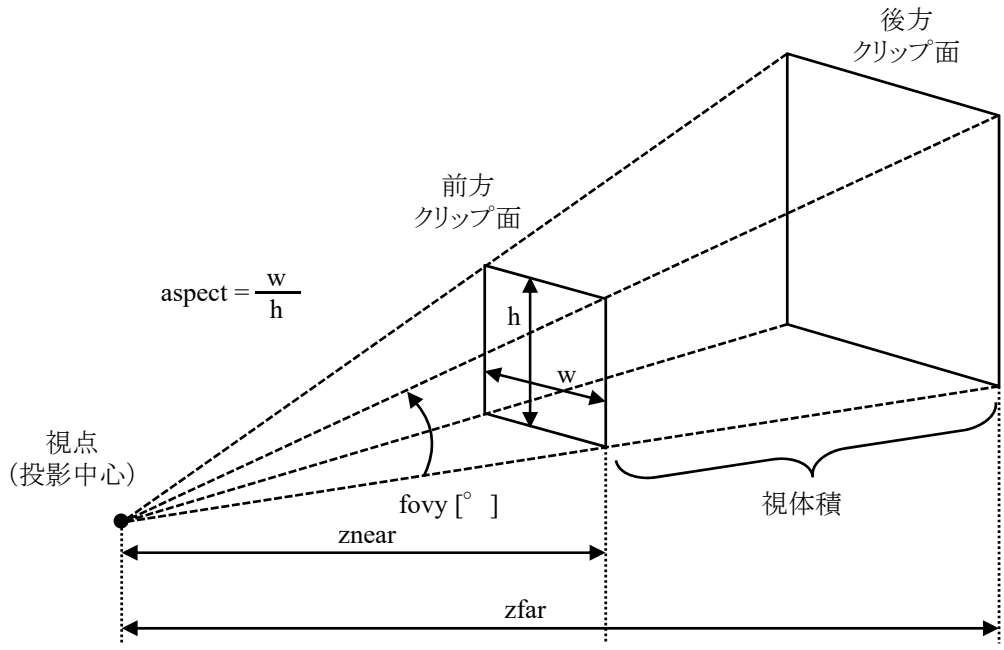
視体積

- 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

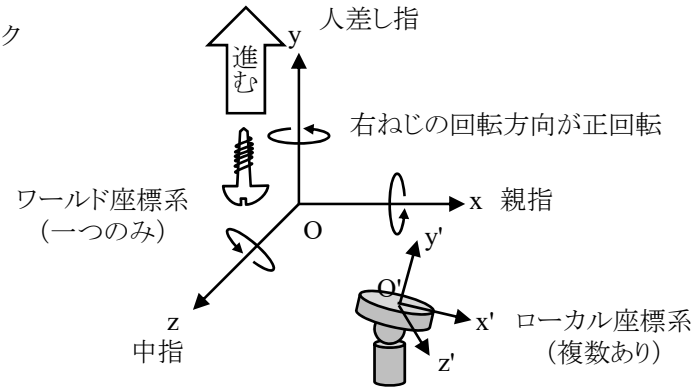
- void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar)
  - 変換行列に透視変換の行列を乗じます．
  - 最初の引数「fovy」はカメラの画角であり，「度」で表します．これが大きいほど広角（透視が強くなり，絵が小さくなります）になり，小さいほど望遠レンズになります．
  - 2つ目の引数「aspect」は画面のアスペクト比（縦横比）であり，「1」であればビューポートに表示される図形のx方向とy方向のスケールが等しくなります．
  - 3つ目の引数「zNear」と4つ目の引数「zFar」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

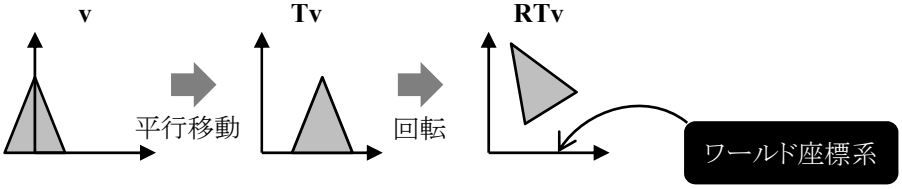
- ワールド座標系：ユーザが描く図形やオブジェクト全てに対して共通な座標系である
- ローカル座標系：各オブジェクト固有の座標系であり，ワールド座標系の中に複数存在する



モデリング変換

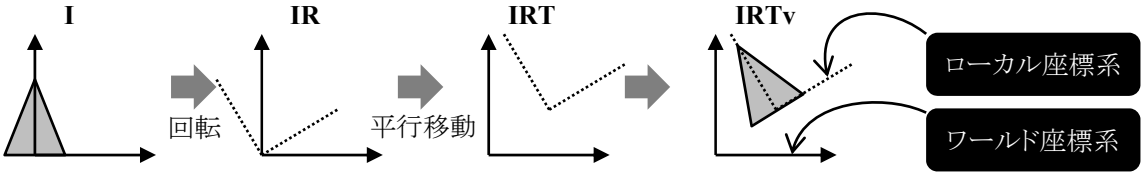
- OpenGLは，オブジェクトの平行移動(translation)，回転(rotation)，スケーリング(scaling)の三つのモデリング変換を用意している
- OpenGLは，2次元や3次元の頂点を扱うが，内部では統一的に4次元ベクトル(x, y, z, w)を用いる．これを同次座標という．
- OpenGLでは，点$\mathbf{v}=(x\ y\ z\ 1)^{\mathrm{T}}$を点$\mathbf{v}'=(x'\ y'\ z'\ 1)^{\mathrm{T}}$に変換する一般式は$\mathbf{v}'=\mathbf{T}\mathbf{v}$で表される
- $\mathbf{T}$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

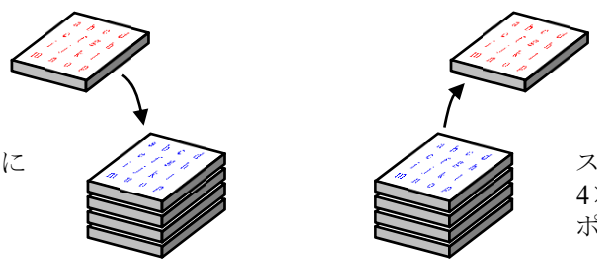
ローカル座標系の考え方



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

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.l);
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();
```

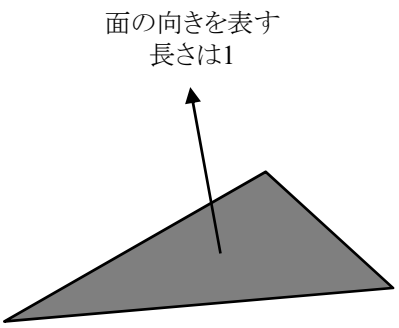
例3
```
glPushMatrix();
glTranslated(posx, posy, 0.1);
glBegin(GL_QUADS);
glVertex3d(-0.5, -0.5, 0.0);
glVertex3d(+0.5, -0.5, 0.0);
glVertex3d(+0.5, +0.5, 0.0);
glVertex3d(-0.5, +0.5, 0.0);
glEnd();
glPopMatrix();
```

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();
```

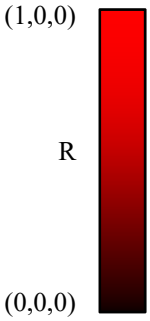


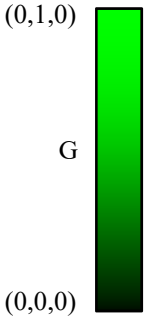
関数の説明

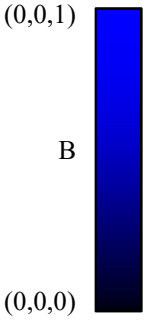
- void glNormal3d(GLdouble nx, GLdouble ny, GLdouble nz);
  - 法線ベクトルを設定します．

色の表現

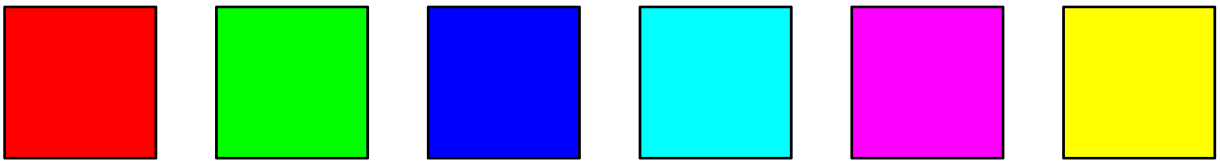
- 光の三原色で表す
- R: Red 赤, G: Green 緑, B: Blue 青
- glClearColor関数で背景色を指定，glColor3d関数で描画する図形の色を指定します
- OpenGLのこれらの関数の引数は，浮動小数点の0～1の値で表します
  - ちなみに，画面や画像ファイルなどの画素は通常，RGBそれぞれ8bitずつ，合計24bitで表します．整数で0～255の値で表します．OpenGLは0～1ですのでご注意ください．
- 0が暗くて，1が明るいです







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



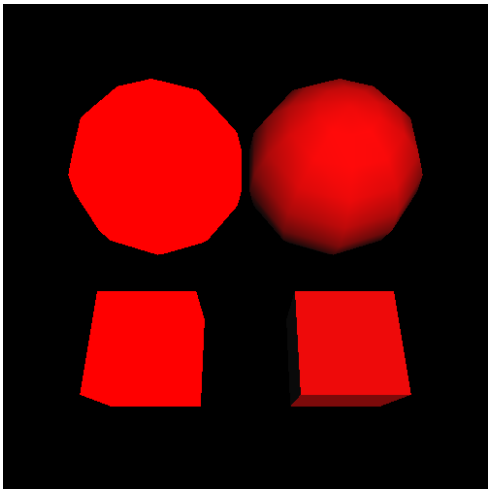
(1,0,0)　(0,1,0)　(0,0,1)　(0,1,1)　(1,0,1)　(1,1,0)

(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

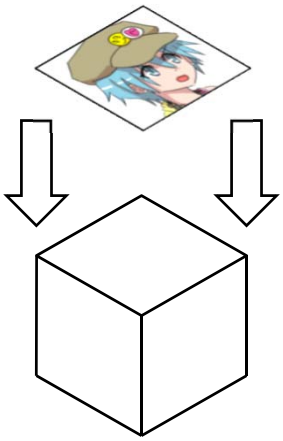
- glEnable(GL_LIGHTING);
  - 光源による陰影を計算する機能をオンにする
- glEnable(GL_LIGHTn);
  - 番号n=0～7の光源を点灯する
- glLightfv(GL_LIGHTn, pname, params);
  - 番号n=0～7の光源の属性を設定する
- glDisable(GL_LIGHTn);
  - 番号n=0～7の光源を消灯する
- glDisable(GL_LIGHTING);
  - 光源による陰影を計算する機能をオフにする

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.

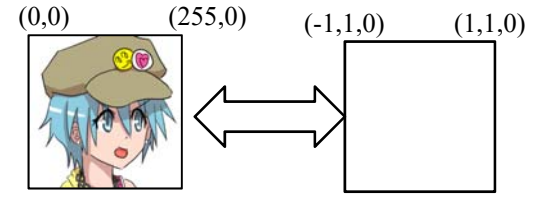
pname	params	パラメータ
GL_DIFFUSE	float型の4次元配列	材質の拡散RGBA値

テクスチャマッピングの座標系

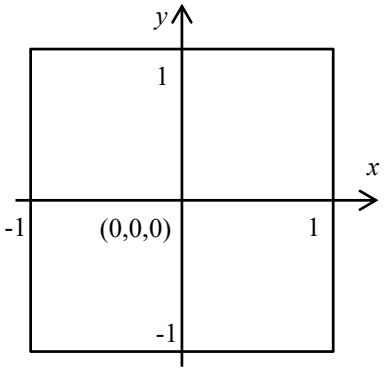


3次元の面に2次元の画像を貼り付ける＝テクスチャマッピング

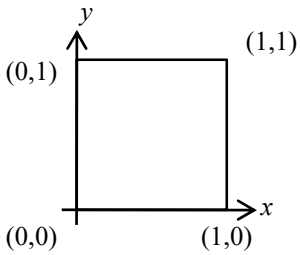
点の座標を対応づける



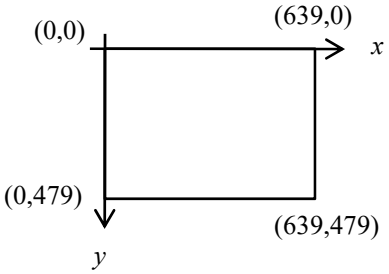
2次元座標　　3次元座標



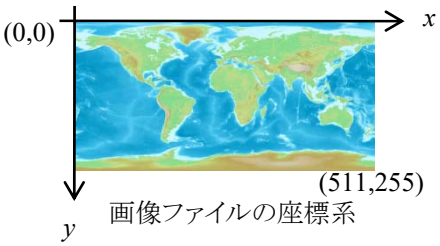
OpenGLの空間の座標系（初期状態）



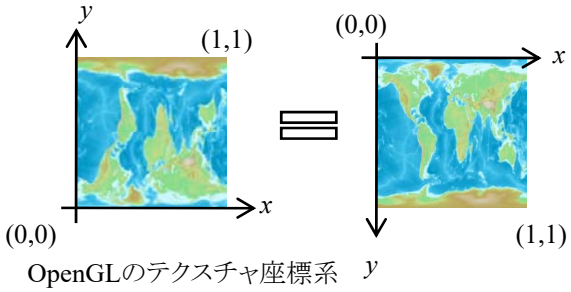
OpenGLのテクスチャ座標系



画像やディスプレイの座標系



画像ファイルの座標系



OpenGLのテクスチャ座標系

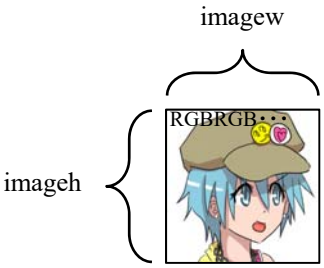
```
glBegin(GL_QUADS);
glTexCoord2d(0.0, 0.0); glVertex3d(-1.0, 1.0, 0.0);
glTexCoord2d(0.0, 1.0); glVertex3d(-1.0, -1.0, 0.0);
glTexCoord2d(1.0, 1.0); glVertex3d(1.0, -1.0, 0.0);
glTexCoord2d(1.0, 0.0); glVertex3d(1.0, 1.0, 0.0);
glEnd();
```

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);
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



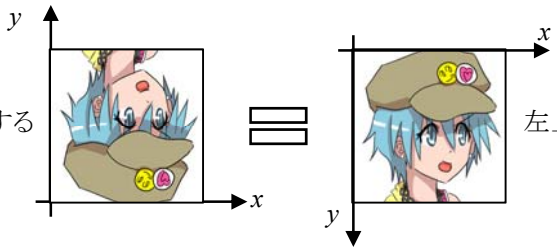
unsigned char
RGBRGB…の順

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



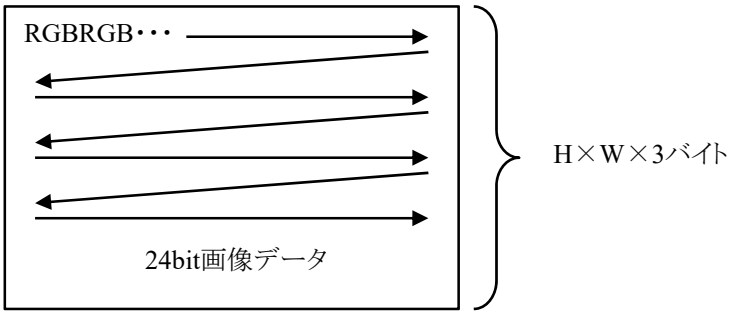
左上が原点だと思うと分かりやすい

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

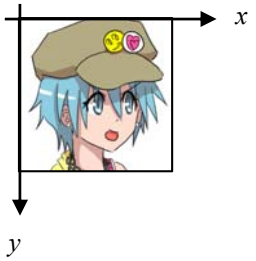
画像ファイル

- 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

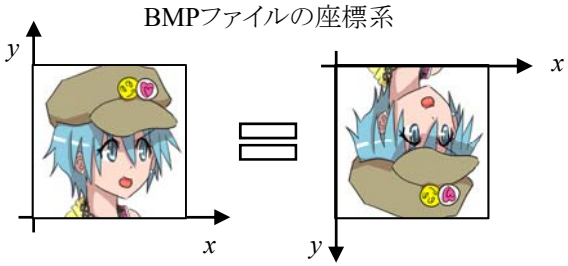


- このままファイルとして表現してしまうと，高さHと幅Wが分からない
- 高さHや幅Wやその他の情報をヘッダとして，ファイルの先頭部分に格納する
- 画像ファイルの種類は様々なものが存在するが，それぞれ，ヘッダの内容や形式，画像データの表現形式，などに違いがある
- Windowsでは標準でBMPファイルが利用されている
- BMP（ビー・エム・ピー）はbitmap（ビットマップ）の略で，拡張子はbmpである
- 圧縮形式と非圧縮形式があるが，圧縮形式は詳細が不明であるので扱わないほうがいい
- 1bit2色，4bit16色，8bit256色の形式もあるが，ここでは24bit1670万色の形式のみを説明する
- 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）が入れられる

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

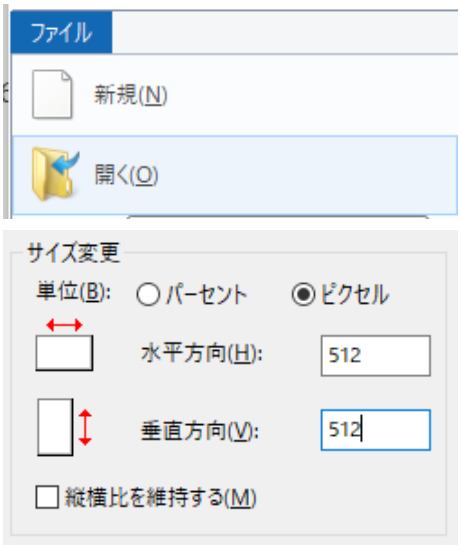
画像の高さと幅は64，128，256，512，1024など

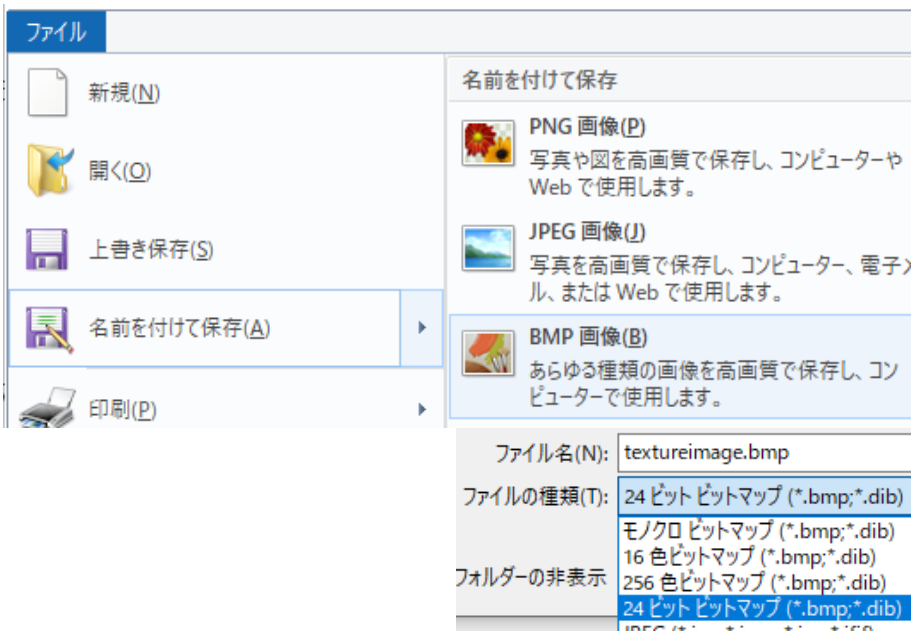
bmpファイル

- 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

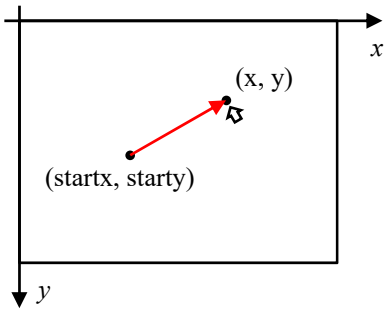
```
void myMouse(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        startx = x;
        starty = y;
        moving = true;
    }
    else {
        /* 省略 */
    }
}
```

左ボタンを押す

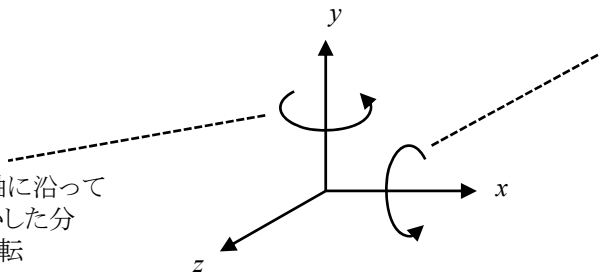
moving = true

```
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



画面上のx軸に沿って
マウスを動かした分
y軸周りに回転

画面上のy軸に沿って
マウスを動かした分
x軸周りに回転

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);
```

左ボタンを放す

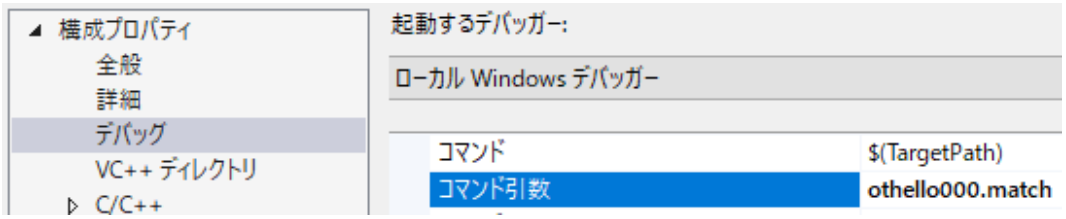
moving = false

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

コマンドライン引数

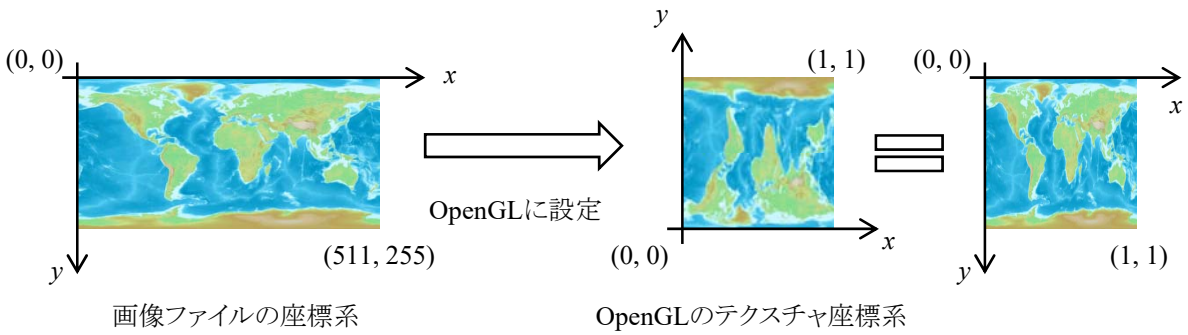
- コマンド引数にファイル名を指定して実行しないといけない
- まず，配布したデータファイル（以降earth.bmpと呼ぶことにする）をcppファイルのあるディレクトリにコピーする

- 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earth.bmpと書く



地球儀へのテクスチャマッピング

- 正距円筒図法で表された以下のようなテクスチャを3次元の球に貼れ

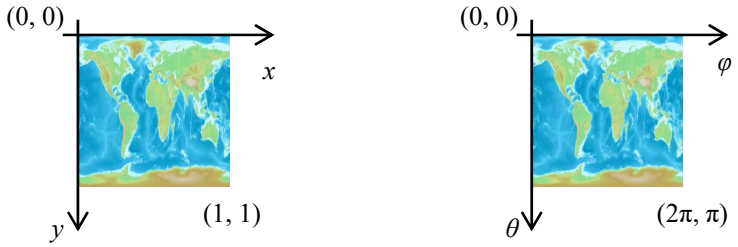


画像ファイルの座標系　　OpenGLのテクスチャ座標系

- 3次元座標はデカルト座標(x,y,z)のほか，極座標(r,θ,φ)で表すこともできる
- rは半径，θは天頂角，φは方位角と呼ぶことにする
- ここでは，極座標系として，θ=0°を北極とし，日付変更線のあたりをφ=0°とし，θは0°～180°，φは0°～360°とし，θは北極から南極へ向かう方向，φは日付変更線から東へアメリカ・ヨーロッパ・アジアを経て再び日付変更線に戻る方向，とする

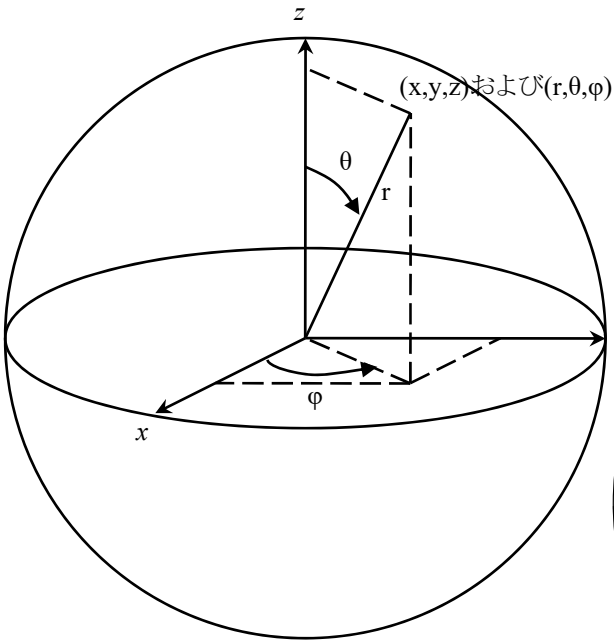
課題プログラムの説明

- myGlobe関数で半径sizeの球を描画する
- 球は四角形の集まりとして表現する．四角形の解像度は整数型の引数resolutionで指定する
- 外側のfor文はφについてのループ，内側のfor文はθについてのループ
  - phiiは0から2*resolution-1までの整数値
  - thetaiは0からresolution-1までの整数値
- C言語のsin関数やcos関数の引数の角度はdegreeではなくradianなので注意
- M_PIは円周率πを表す
- 整数型の値と浮動小数点型の値が混在する計算を行うときは，型の変換を正しく行うように十分注意すること



OpenGLのテクスチャ座標系　　テクスチャの極座標系

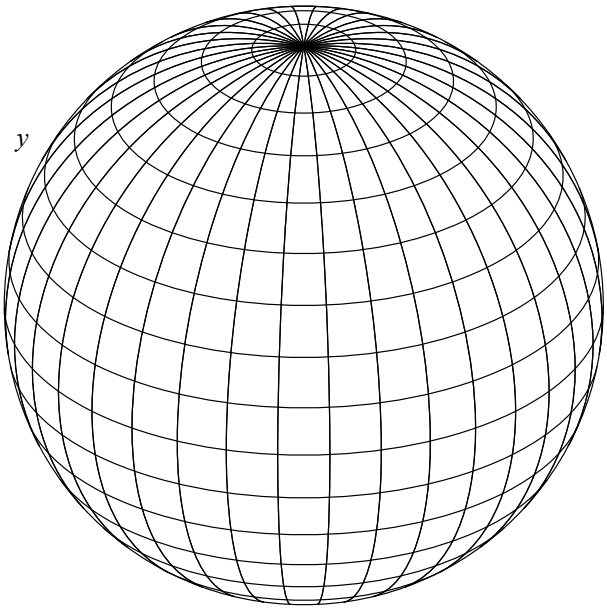
地球儀へのテクスチャマッピング



球の極座標系

3次元の極座標系とデカルト座標系の対応関係

$$x = r\sin\theta\cos\phi$$
$$y = r\sin\theta\sin\phi$$
$$z = r\cos\theta$$

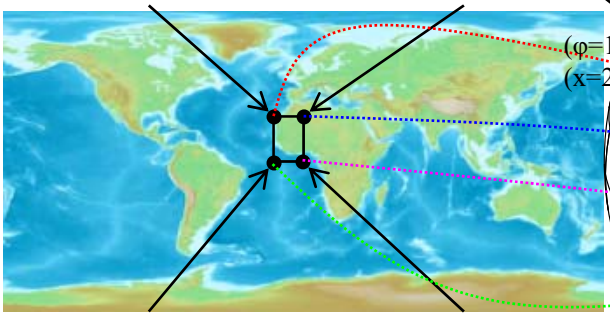


球のデータの離散的表現（天頂角と方位角を均等割）

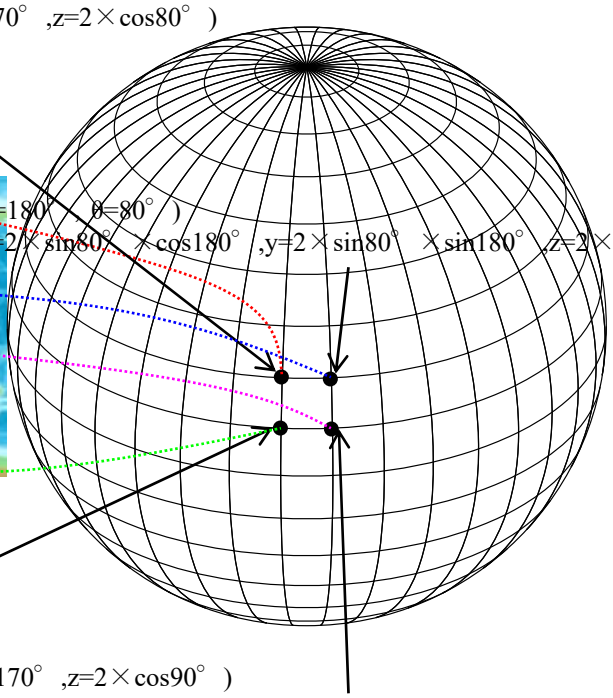
(φ=170°，θ=80°)
(x=2×sin80° ×cos170°,y=2×sin80° ×sin170°,z=2×cos80°)

(φ=170°，θ=80°)
(u=170/360,v=80/180)

(φ=180°，θ=80°)
(u=180/360,v=80/180)



(φ=180°，θ=80°)
(x=2×sin80° ×cos180°,y=2×sin80° ×sin180°,z=2×cos80°)



(φ=170°，θ=90°)
(u=170/360,v=90/180)

(φ=180°，θ=90°)
(u=180/360,v=90/180)

ただし，(u,v)はテクスチャ座標系を表す

(φ=170°，θ=90°)
(x=2×sin90° ×cos170°,y=2×sin90° ×sin170°,z=2×cos90°)

(φ=180°，θ=90°)
(x=2×sin90° ×cos180°,y=2×sin90° ×sin180°,z=2×cos90°)

各四角形における，2次元と3次元の座標の対応関係の例

※半径が2で，θとφの離散化の間隔を10°とした場合の例

プロトタイプ

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

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

void myGlobe(double size, int resolution)
{
    int phii, thetai;
    //［課題］変数の宣言(もし必要であれば)

    glBegin(GL_QUADS);
    for (phii = 0; phii < 2 * resolution; phii++) {
        for (thetai = 0; thetai < resolution; thetai++) {
            //［課題］途中計算(もし必要であれば)

            //［課題］glTexCoord2dに1番目の頂点のテクスチャ座標を指定する
            //［課題］glVertex3dに1番目の頂点のデカルト座標を指定する

            //［課題］glTexCoord2dに2番目の頂点のテクスチャ座標を指定する
            //［課題］glVertex3dに2番目の頂点のデカルト座標を指定する

            //［課題］glTexCoord2dに3番目の頂点のテクスチャ座標を指定する
            //［課題］glVertex3dに3番目の頂点のデカルト座標を指定する

            //［課題］glTexCoord2dに4番目の頂点のテクスチャ座標を指定する
            //［課題］glVertex3dに4番目の頂点のデカルト座標を指定する

            //［課題］ここから
            glTexCoord2d(170.0 / 360.0, 80.0 / 180.0);
            glVertex3d(2.0 * sin(80.0 / 180.0 * M_PI) * cos(170.0 / 180.0 * M_PI),
                2.0 * sin(80.0 / 180.0 * M_PI) * sin(170.0 / 180.0 * M_PI),
                2.0 * cos(80.0 / 180.0 * M_PI));
            glTexCoord2d(170.0 / 360.0, 90.0 / 180.0);
            glVertex3d(2.0 * sin(90.0 / 180.0 * M_PI) * cos(170.0 / 180.0 * M_PI),
                2.0 * sin(90.0 / 180.0 * M_PI) * sin(170.0 / 180.0 * M_PI),
                2.0 * cos(90.0 / 180.0 * M_PI));
            glTexCoord2d(180.0 / 360.0, 90.0 / 180.0);
            glVertex3d(2.0 * sin(90.0 / 180.0 * M_PI) * cos(180.0 / 180.0 * M_PI),
                2.0 * sin(90.0 / 180.0 * M_PI) * sin(180.0 / 180.0 * M_PI),
                2.0 * cos(90.0 / 180.0 * M_PI));
            glTexCoord2d(180.0 / 360.0, 80.0 / 180.0);
            glVertex3d(2.0 * sin(80.0 / 180.0 * M_PI) * cos(180.0 / 180.0 * M_PI),
                2.0 * sin(80.0 / 180.0 * M_PI) * sin(180.0 / 180.0 * M_PI),
                2.0 * cos(80.0 / 180.0 * M_PI));
            //［課題］ここまで削除
        }
    }
    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);
    glTexEnvi(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);

    myGlobe(2.0, 18);

    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);
}
```

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[])
{
    imagedata = NULL;
    anglex = 0.0;
    angley = 0.0;
    moving = false;
    if (argc <= 1) {
        printf("filename not specified¥n");
        printf("push any key and push enter¥n");
        return 1;
    }
    if (readbmp(argv[1], &imagedata, &imagew, &imageh)) {
        printf("push any key and push enter¥n");
        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！

バーチャルトラックボール

- 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「サイコロ」の課題に載っているバーチャルトラックボールを実装しましょう