

- 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 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 glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)
 - 5 「glClearColor(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「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。
- 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 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 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_LIGHT1)」を実行すると、1番目の光源を点灯します。
 - 5 「glEnable(GL_BLEND)」を実行すると、ブレンド処理が有効化します。
 - 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 「glEnable(GL_ALPHA_TEST)」を実行すると、それ以降「アルファ・テスト」をするようになります。
- 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。

- 5 `void glutDisplayFunc(void (*)(void))`
 - 5 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。
- 5 `void glutReshapeFunc(void (*)(int width, int height))`
 - 5 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。
- 5 `void glutTimerFunc(unsigned int millis, void (*)(int value), int value)`
 - 5 指定された時間に呼び出されるコールバック関数を登録します。異なる時間のコールバック関数を複数用意できます。
 - 5 「`millis`」は呼び出される時間をミリ秒で指定します。少なくとも「`millis`」ミリ秒後にコールされるようになります。
 - 5 第3引数の「`value`」は登録したタイマーコールバック関数に渡されます。
- 5 `void glutKeyboardFunc(void (*)(unsigned char key, int x, int y))`
 - 5 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。
- 5 `void glutSpecialFunc(void (*)(int key, int x, int y))`
 - 5 特殊キーが押されたときに呼び出す関数を指定します。`key`には押された特殊キーのコードが渡されます。
 - 5 矢印キーの場合は`GLUT_KEY_LEFT`, `GLUT_KEY_RIGHT`, `GLUT_KEY_UP`, `GLUT_KEY_DOWN`です。
- 5 `void glutMouseFunc(void (*)(int button, int state, int x, int y))`
 - 5 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。
- 5 `void glutMotionFunc(void (*)(int x, int y))`
 - 5 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。
 - 5 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)。
- 5 `void glutPostRedisplay(void)`
 - 5 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。
- 5 `void glViewport(GLint x, GLint y, GLsizei width, GLsizei height)`
 - 5 「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 `glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble zNear, GLdouble zFar)`
 - 5 正投影の投影行列を作成し、それを現在の行列に乗算します。
 - 5 視体積は箱形で、その8つの点は(left, top, zNear), (left, bottom, zNear), (right, bottom, zNear), (right, top, zNear), (left, top, zFar), (left, bottom, zFar), (right, bottom, zFar), (right, top, zFar)です。
 - 5 2次元図形しか描画しない場合は通常、zNear=-1, zFar=1を指定します。
- 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 `gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz)`
 - 5 この最初の3つの引数「eyex, eyey, eyez」は視点の位置、次の3つの引数「centerx, centery, centerz」は目標の位置、最後の3つの引数「upx, upy, upz」は、ウィンドウに表示される画像の「上」の方向を示すベクトルです。
- 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 `glScaled(GLdouble x, GLdouble y, GLdouble z)`
 - 5 変換行列に拡大縮小の行列を乗じます。3つの引数「x」「y」「z」には拡大係数を指定します
 - 5 これで拡大縮小すると、法線ベクトルがズレて陰影付けがおかしくなってしまいます
- 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 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 glVertex2d(GLdouble x, GLdouble y)
void glVertex3d(GLdouble x, GLdouble y, GLdouble z)
void glVertex3dv(GLdouble *v)
 - 5 2次元、または、3次元の座標値を設定します。
 - 5 引数は、2次元座標の場合はGLdouble型の(x, y), 3次元は(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 glColor3d(GLdouble red, GLdouble green, GLdouble blue)
void glColor4d(GLdouble red, GLdouble green, GLdouble blue, GLdouble alpha)
 - 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 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)。
- 5 void glMaterialfv(GLenum face, GLenum pname, GLfloat *params)
void glMaterialf(GLenum face, GLenum pname, GLfloat param)
 - 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。
 - 5 「pname」に「GL_SPECULAR」を指定すると、「params」でfloat型の配列を指定することで、材質の鏡面RGBA値を設定できます。その際はglMaterialfvを使います。
 - 5 「pname」に「GL_SHININESS」を指定すると、「param」でfloat型の数値を指定することで、鏡面係数を設定できます。その際はglMaterialfを使います。
- 5 void glLightfv(GLenum light, GLenum pname, GLfloat *params)
 - 5 光源の属性と位置などを設定します
 - 5 lightには設定する光源の番号に応じてGL_LIGHT0~GL_LIGHT7のいずれかを指定します
 - 5 pnameにGL_POSITIONを指定すると、paramsでfloat型の配列を指定することで、光源の位置(x,y,z,w)を設定できます。点光源はw=1として(x,y,z)で座標を指定します。平行光源はw=0として(x,y,z)で方向ベクトルを指定します。
 - 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、拡散光のRGBA値を設定できます。
 - 5 pnameにGL_SPECULARを指定すると、paramsでfloat型の配列を指定することで、鏡面光のRGBA値を設定できます。

- 5 void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks)
 - 5 GLUTが提供する3D基本オブジェクトの1つ。球。球の中心は、初期値では原点にある。
 - 5 radius 球の半径
 - 5 slices 緯度方向の分割数
 - 5 stacks 経度方向の分割数
- 5 void glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings)
 - 5 GLUTが提供する3D基本オブジェクトの1つ。トーラス(ドーナツ状の形)。その中心と方向は、初期値では原点およびz軸方向にある。すなわち、トーラスの中心軸はz軸と平行となる。
 - 5 innerRadius トーラスの内半径
 - 5 outerRadius トーラスの外半径
 - 5 sides トーラスを円筒で分割したとき、円筒の表面の分割数
 - 5 rings トーラスを円筒での分割数
- 5 void glutSolidOctahedron(void)
 - 5 8面体。中心は原点、半径は1
- 5 void glGetDoublev(GLenum pname, GLdouble *params)
 5 void glGetIntegerv(GLenum pname, GLint *params)
 - 5 状態変数を取得します。引数pnameは、返すべき状態変数を示すシンボル定数で、paramsは、返されたデータが配置される、指示した型式による配列を指すポインタです。
 - 5 GL_MODELVIEW_MATRIX モデルビュー行列スタック
 - 5 GL_PROJECTION_MATRIX 射影行列スタック
 - 5 GL_VIEWPORT ビューポートの原点と範囲
- 5 int gluProject(GLdouble objx, GLdouble objy, GLdouble objz, const GLdouble modelMatrix[16], const GLdouble projMatrix[16], const GLint viewport[4], GLdouble *winx, GLdouble *winy, GLdouble *winz)
 - 5 モデルビュー行列(modelMatrix)、射影行列(projMatrix)、ビューポート(viewport)が定義する変換を使用して、指定したオブジェクト座標(objx, objy, objz)をウィンドウ座標に写像します。その結果のウィンドウ座標はwinx, winy, winzに返されます。
- 5 void glBlendFunc(GLenum sfactor, GLenum dfactor)
 - 5 ブレンディング係数を設定します。
 - 5 sfactorには、source側のブレンディング係数を指定します。初期値はGL_ONEです。
 - 5 dfactorには、destination側のブレンディング係数を指定します。初期値はGL_ZEROです。
 - 5 アルファブレンディングを設定するには、sfactorとしてGL_SRC_ALPHA、dfactorとしてGL_ONE_MINUS_SRC_ALPHAを設定します。
 - 5 そのときのブレンディングの方程式は

$$\text{output} = \text{source} * A_s + \text{destination} * (1 - A_s)$$
 という式になります。ここで、 A_s はsource側のA値(アルファ値)です。
- 5 void glAlphaFunc(GLenum func, GLclampf ref)
 - 5 α 値の比較関数を設定します。アルファ・テスト用です。
 - 5 funcにGL_ALWAYSを設定した場合はアルファ・テストは特に何もしません。
 - 5 つまり、GL_ALPHA_TESTとしては不透明と見なして処理します(GL_ALPHA_TESTが不透明として扱っているだけであり、GL_BLENDやGL_STENCIL_TESTが透明や不透明として扱うかどうかとは無関係です)
 - 5 funcにGL_GREATERを設定して、refに0.5を設定した場合
 - 5 α 値が1のように0.5より大きい値の場合は不透明と見なして処理します。
 - 5 α 値が0のように0.5より小さい値の場合は透明と見なして処理します。

OpenGL

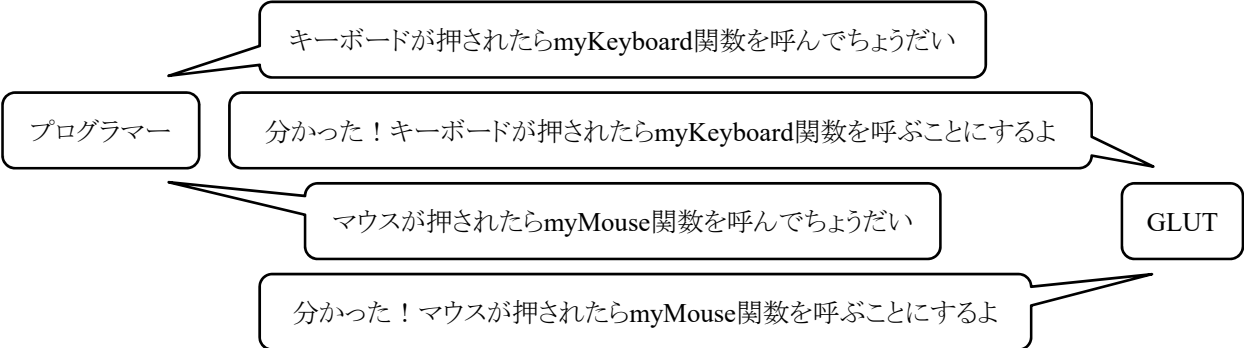
- ❏ `void glRasterPos2d(GLdouble x, GLdouble y)`
`void glRasterPos3d(GLdouble x, GLdouble y, GLdouble z)`
 - ❏ ラスタ位置を決定する. 位置は, 現在のモデルビュー行列あるいは投影行列を用いて, スクリーン座標系に射影される.
- ❏ `void glutBitmapCharacter(void *font, int character)`
 - ❏ `font`で指定されたフォントを用いて, ASCIIコードで与えられた文字`character`を描く. `font`には `GLUT_BITMAP_TIMES_ROMAN_24`などがある.
- ❏ `int glutBitmapWidth(void *font, int character)`
 - ❏ `font`で指定したフォントにおける文字`character`の幅を返す. 幅の単位は画素.
- ❏ `void glPushAttrib(GLbitfield mask)`
`void glPopAttrib(void)`
 - ❏ `glPushAttrib()`は, `mask`のビットが示す全属性を属性スタック上にプッシュすることにより, それらを保存します. `glPopAttrib()`は, 最後の`glPushAttrib()`で保存された状態変数の値をリストアします.
 - ❏ `GL_LINE_BIT` 線
 - ❏ OpenGLでは, 関連する状態変数を属性グループにグループ分けします. 例えば, 属性 `GL_LINE_BIT`は, 線幅, `GL_LINE_STIPPLE`の有効ステータス, 破線パターン, 破線の反復数, `GL_LINE_SMOOTH`の有効ステータスという5つの状態変数から構成されています. コマンド`glPushAttrib()`と`glPopAttrib()`を使用すると, 5つの状態変数すべてを一度に保存, リストアできます.
- ❏ `void glLineWidth(GLfloat width)`
 - ❏ レンダリングする線の幅をピクセルで設定します. `width`は0.0より大きくする必要があり, 初期設定では1.0になっています.
- ❏ `void glutStrokeCharacter(void *font, int character)`
 - ❏ `font`により指定されたストロークフォントで, ASCIIコードで与えられた文字`character`を描く. `font`には, `GLUT_STROKE_ROMAN`と`GLUT_STROKE_MONO_ROMAN`がある.
- ❏ `int glutStrokeWidth(void *font, int character)`
 - ❏ フォント`font`における文字`character`の幅を返す. 幅の単位は画素.

- void glTexCoord2d(GLdouble s, GLdouble t)
void glTexCoord2dv(GLdouble *v)
 - 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 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せん。物体の材質やライトによる陰影の影響を受けません
- 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
 - 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

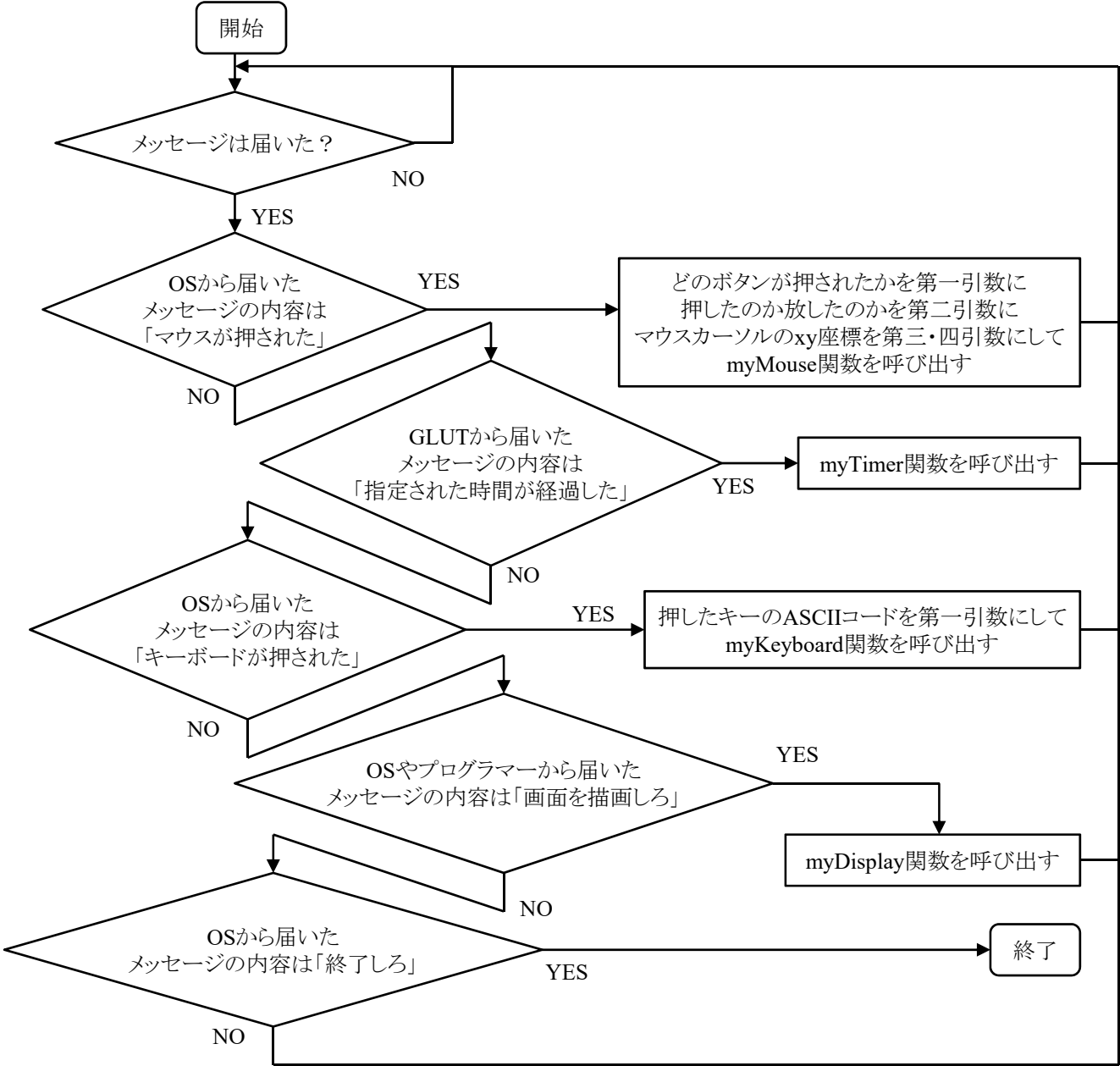
イベントドリブン型プログラミング

- 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



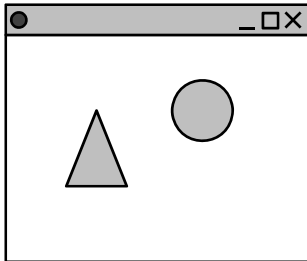
ループ中の動作の例



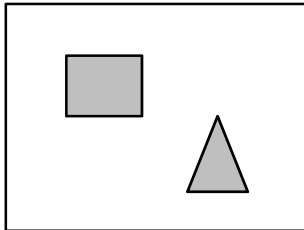
画面の描画

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

描画の流れ

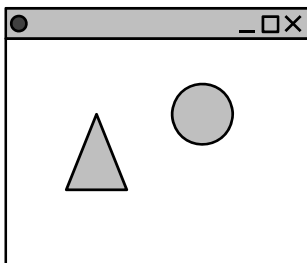


ウィンドウ表示用のバッファ



メモリ内にあるバッファ

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

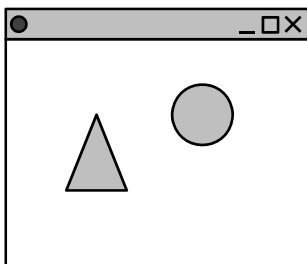


ウィンドウ表示用のバッファ

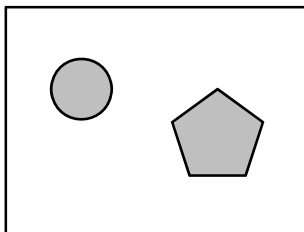


メモリ内にあるバッファ

```
glBegin~glEnd  
図形の描画
```

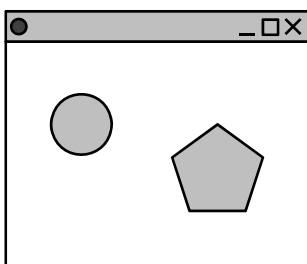


ウィンドウ表示用のバッファ

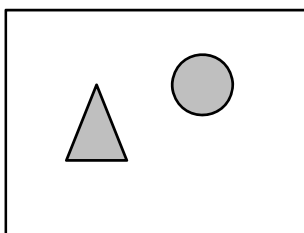


メモリ内にあるバッファ

```
glutSwapBuffers();  
ウィンドウに表示
```



ウィンドウ表示用のバッファ



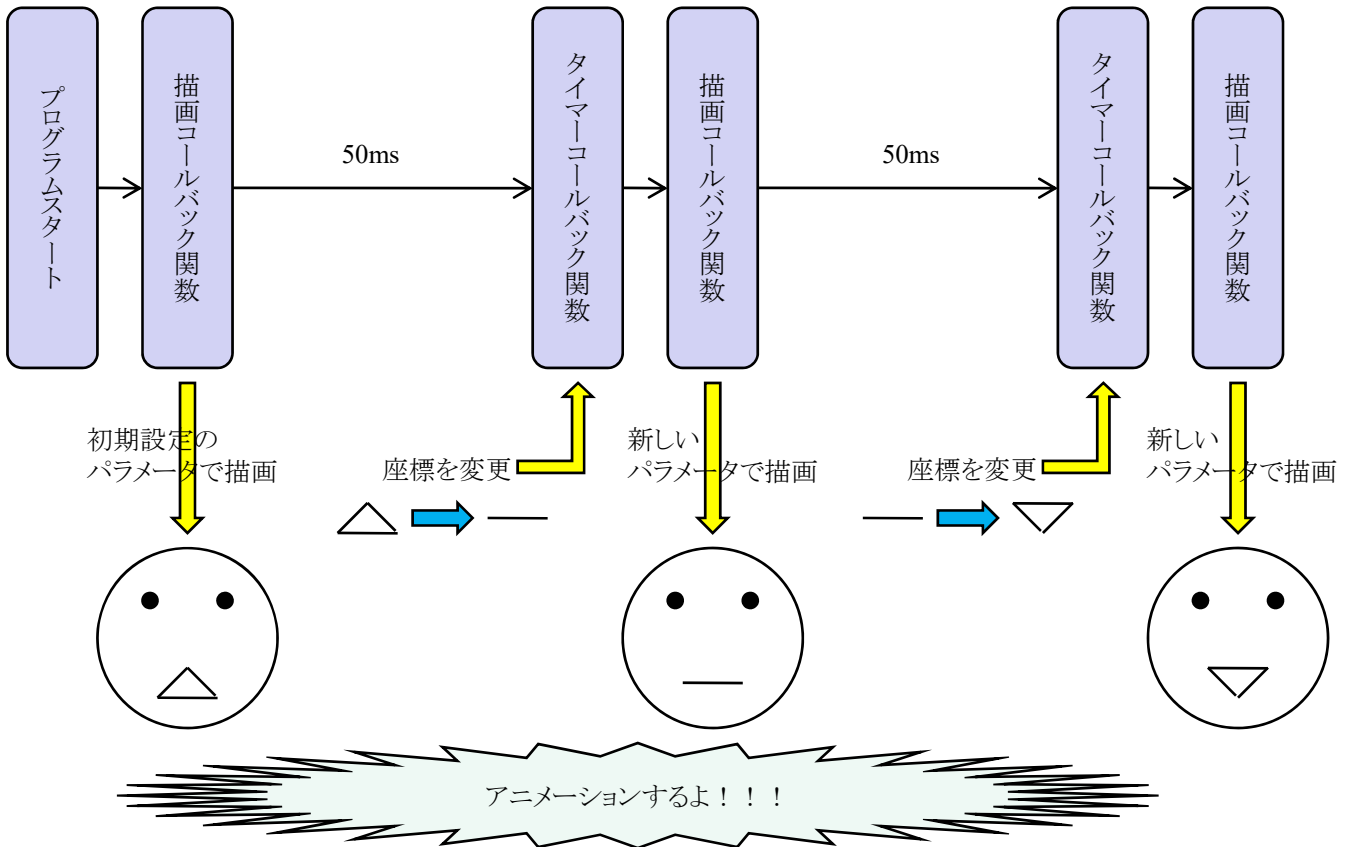
メモリ内にあるバッファ

```
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();  
}
```

タイマー

- void glutTimerFunc(unsigned int msec, void (*func)(int value), int value)
 - 指定された時間に呼び出されるコールバック関数を登録します。異なる時間のコールバック関数を複数用意できます。
 - 「msec」は呼び出される時間をミリ秒で指定します。少なくとも「msec」ミリ秒後にコールされるようになります。
 - 第3引数の「value」は登録したタイマーコールバック関数に渡されます。
- void glutPostRedisplay(void)
 - ウィンドウ内の再描画を行います。より正確には、現在のウィンドウをマークして、「display関数」を呼び出します。

```
glutTimerFunc(50,myTimer,1); // 50ミリ秒後にmyTimer関数を呼べ
void myTimer(int value) {
    // 描画オブジェクトの座標や各種パラメータを変更する処理を行う
    glutTimerFunc(50,myTimer,1); // 再び50ミリ秒後にmyTimer関数を呼べ
    glutPostRedisplay(); // ウィンドウの内容を再描画しろ(新しい座標やパラメータで描画)
}
```



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();
```

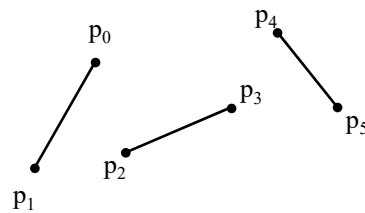
modeは描く図形の種類を指定し, p0, p1, ..., pnは座標位置を意味する.

modeの種類

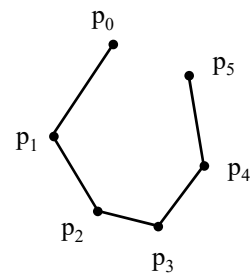
- GL_LINE_STRIP 最初の頂点から最後の頂点まで線分を連結して描画する.
- GL_LINES 二つの頂点を結んだ直線を生成する.
- GL_TRIANGLES 三つ一組の頂点を, それぞれ独立した三角形として描画する.
- GL_QUADS 四つ一組の頂点を, それぞれ独立した四角形として描画する.
- GL_POLYGON 単独の凸ポリゴンを描画する.

ポリゴン描写の注意点

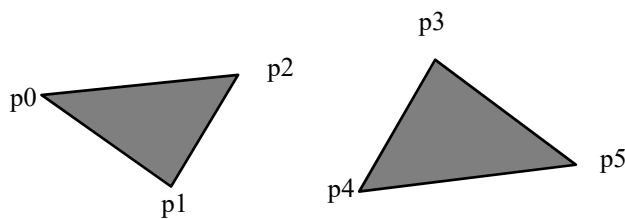
- ポリゴン (polygon) とは多角形の意味
- 頂点座標は反時計回りに設定すること
 - ポリゴンの表面と裏面を区別するため
 - 視点から見て頂点座標が反時計回りに配置されているポリゴンは表面, 時計回りの場合は裏面と約束されている



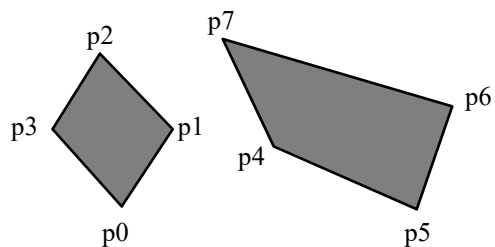
GL_LINES



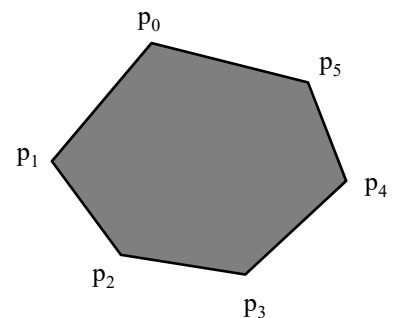
GL_LINE_STRIP



GL_TRIANGLES



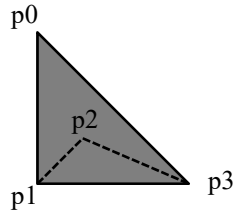
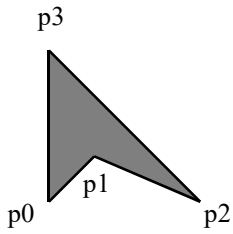
GL_QUADS



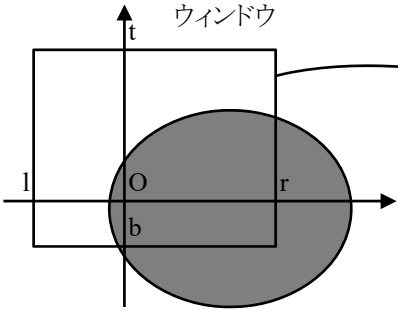
GL_POLYGON

凸多角形

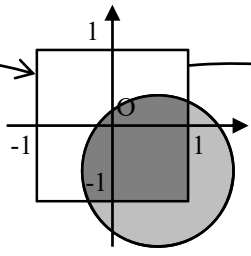
- 凹形状は動作保証対象外
- OpenGLの仕様とは異なる使い方をした場合(この場合, 凸多角形にしか対応していないという仕様を無視して, 非凸多角形を描画しようとした場合)の動作は保証されない.
<http://monobook.org/wiki/OpenGL>
- 通常, 凹多角形は三角形に分割して, 凸多角形で表現するのが普通



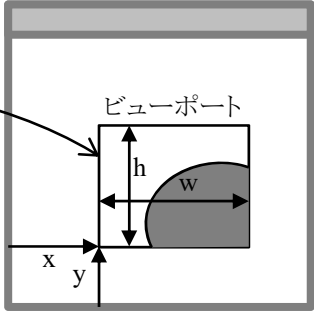
ウィンドウとビューポート



ワールド座標系



正規化デバイス座標系



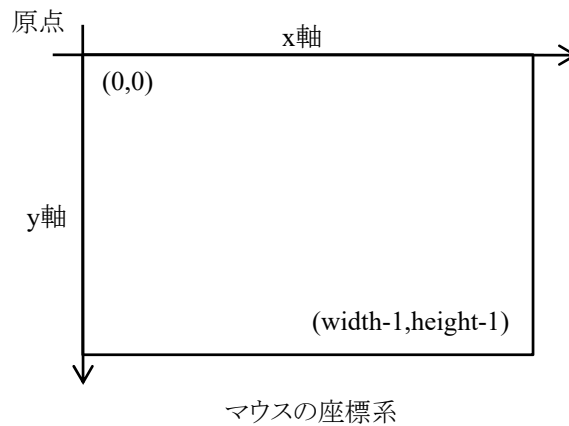
デバイス座標系(ウィンドウ)
(左下が原点)

マウス

- 5 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
- 5 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

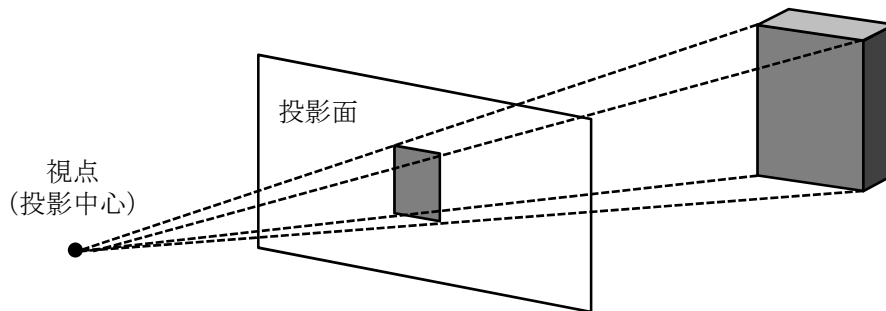
```
コールバック関数の設定  
glutMouseFunc(myMouse);  
glutMotionFunc(myMotion);
```

```
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



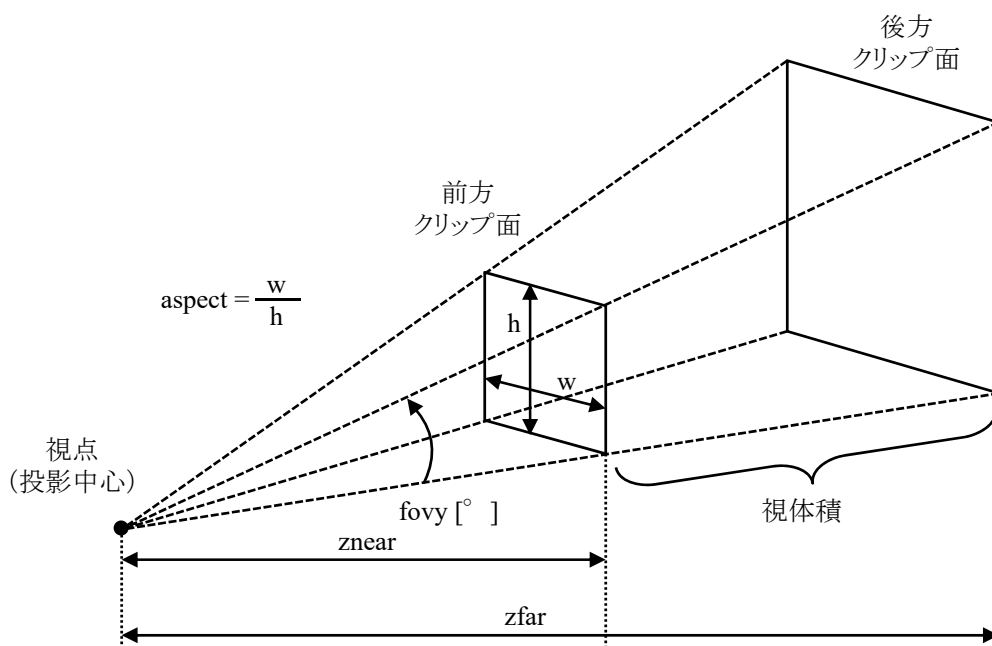
視体積

- 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

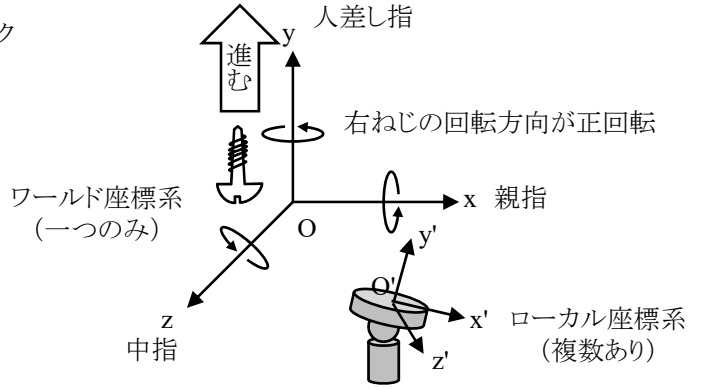
- `void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar)`
 - 変換行列に透視変換の行列を乗じます。
 - 最初の引数「fovy」はカメラの画角であり、「度」で表します。これが大きいほど広角(透視が強くなり、絵が小さくなります)になり、小さいほど望遠レンズになります。
 - 2つ目の引数「aspect」は画面のアスペクト比(縦横比)であり、「1」であればビューポートに表示される図形のx方向とy方向のスケールが等しくなります。
 - 3つ目の引数「zNear」と4つ目の引数「zFar」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

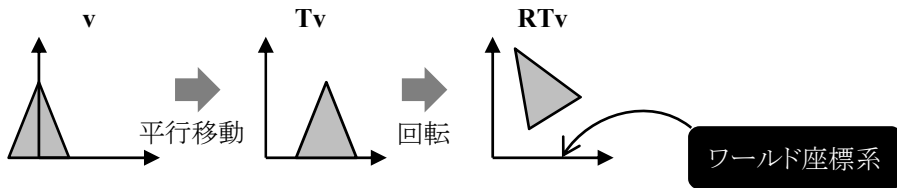
- 📌 ワールド座標系: ユーザが描く図形やオブジェクト全てに対して共通な座標系である
- 📌 ローカル座標系: 各オブジェクト固有の座標系であり, ワールド座標系の中に複数存在する



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 $v=(x\ y\ z\ 1)^T$ 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 4×4 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

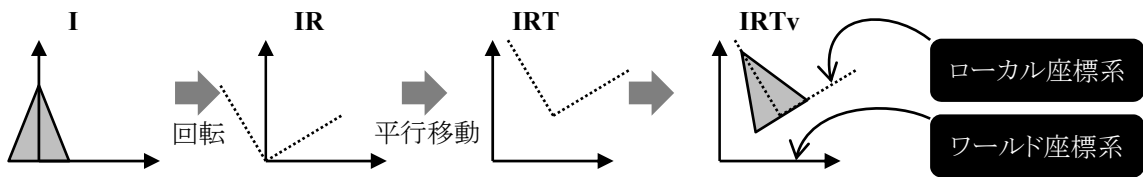
ワールド座標系の考え方



```

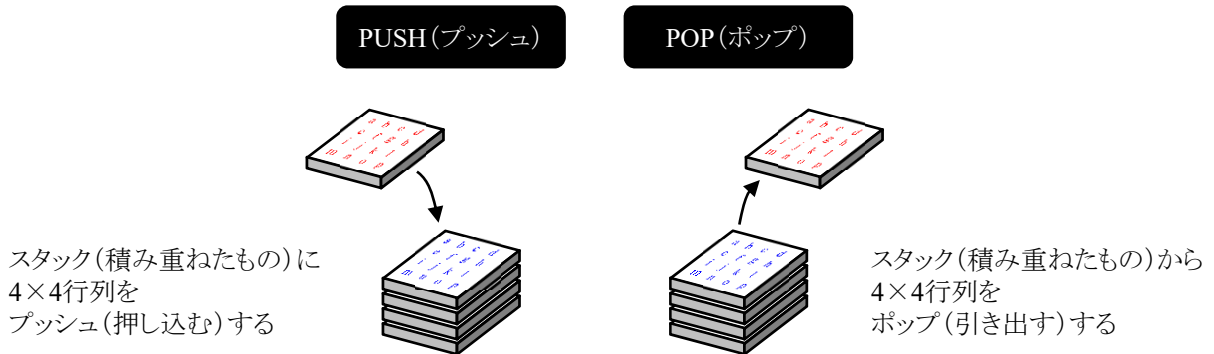
glLoadIdentity();
glRotate*();
glTranslate*();
glWireCone();
    
```

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

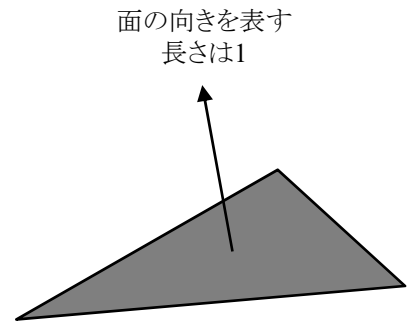


法線

- 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();
```



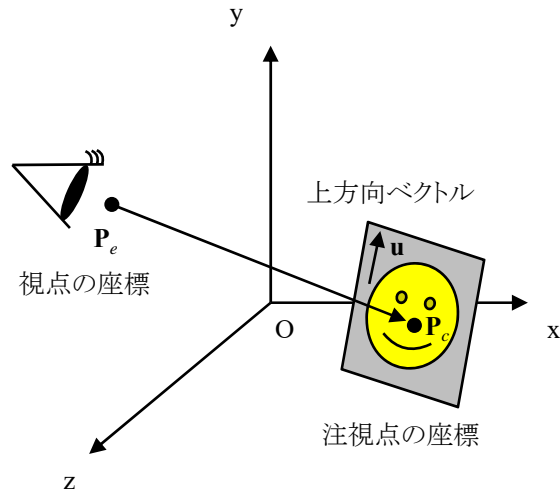
関数の説明

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

視点変換

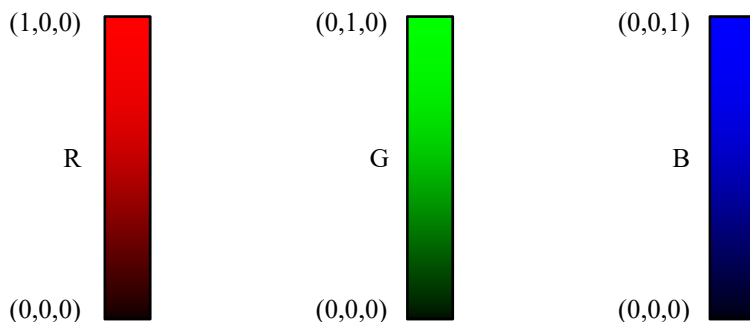
void gluLookAt(GLdouble ex, GLdouble ey, GLdouble ez, GLdouble cx, GLdouble cy, GLdouble cz, GLdouble ux, GLdouble uy, GLdouble uz)

- この最初の3つの引数「ex, ey, ez」は視点の位置, 次の3つの引数「cx, cy, cz」は目標の位置, 最後の3つの引数「ux, uy, uz」は, ウィンドウに表示される画像の「上」の方向を示すベクトルです.



色の表現

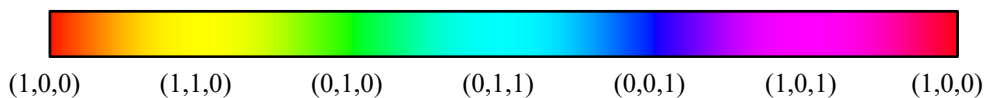
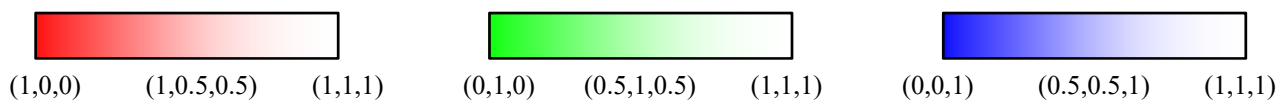
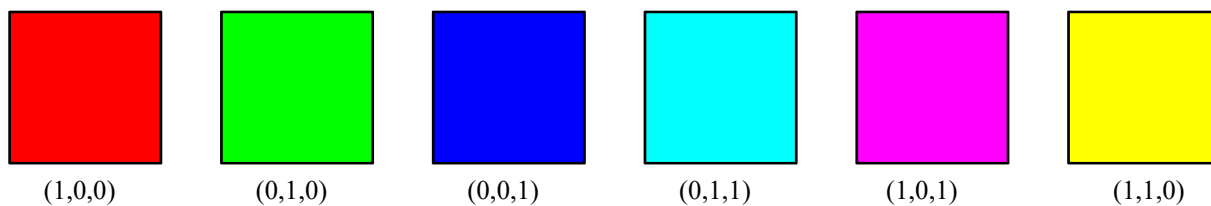
- 光の三原色で表す
- R: Red 赤, G: Green 緑, B: Blue 青
- glClearColor関数で背景色を指定, glColor3d関数で描画する図形の色を指定します
- OpenGLのこれらの関数の引数は, 浮動小数点の0~1の値で表します
 - ちなみに, 画面や画像ファイルなどの画素は通常, RGBそれぞれ8bitずつ, 合計24bitで表します. 整数で0~255の値で表します. OpenGLは0~1ですのでご注意ください.
- 0が暗くて, 1が明るいです



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

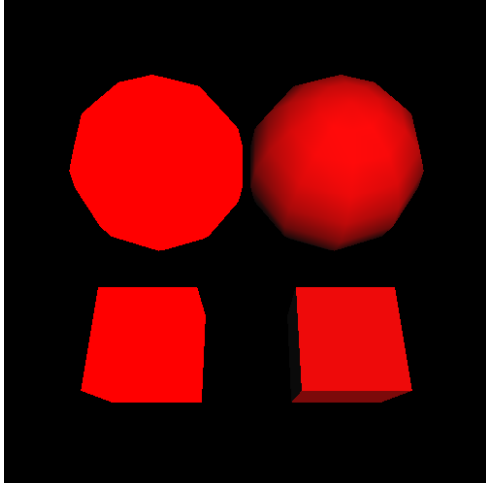
(R,G,B)=(0.5,0.5,0.5)

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



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();
```

光源の種類

❏ 環境光(ambient)

- ❏ 光の方向が特定できないもので、あらゆる方向から発しているように見える。物体への光源は、光源からの直接光の他に、空気による散乱光や他の物体表面からの2次反射、3次反射により、あらゆる方向から弱く照らされている。このため、オブジェクトの影の部分もある程度明るく見える。環境光反射成分を考慮しないと、宇宙空間で撮影した写真のように影の部分が真っ黒となる。環境光が面に当たると、全方向に均等に散乱する。
- ❏ OpenGLでは、環境光を厳密に計算すると膨大になりすぎるため、一定の値でそのオブジェクトを照らすことにより、計算量の低減を図っている。

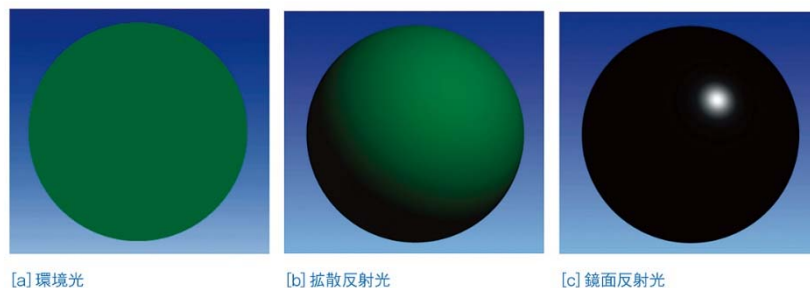
❏ 拡散光(diffuse)

- ❏ 特定の位置・方向から来る光で、光源の成分としても使われる。しかし、一度面に当たると全方向に均等に散乱するため、布地や人肌のようなオブジェクト表面のざらつき度を表現することができる。特定の位置・方向から来る光は全て拡散光を有する。

❏ 鏡面光(specular)

- ❏ 特定の方向から来る光で、面での反射は特定の方向に向かう。高品質の鏡に反射するレーザービームはほとんど鏡面反射となる。磨かれた金属・プラスチックは鏡面光の成分が大きく、チョーク・絨毯などはほとんど持たない。鏡面度は光沢の程度と理解しても差し支えない。

■ 図 4.23——環境光による反射，拡散反射，鏡面反射の各成分



「コンピュータグラフィックス 改訂新版」2015年 / 公益財団法人 画像情報教育振興協会 (CG-ARTS協会)

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
 - ❏ glLightModeli(GL_LIGHT_MODEL_COLOR_MATERIAL, GL_FALSE);
 - ❏ 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

光源の設定

- ❏ void glLightfv(GLenum light, GLenum pname, GLfloat *params)
 - ❏ 光源の属性と位置などを設定します
 - ❏ lightには設定する光源の番号に応じてGL_LIGHT0～GL_LIGHT7のいずれかを指定します

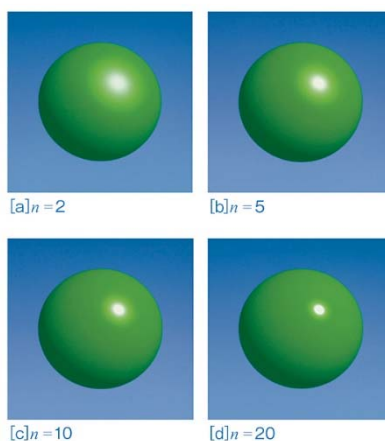
pname	params	パラメータ
GL_AMBIENT	float型の4次元配列	環境光のRGBA値
GL_DIFFUSE	float型の4次元配列	拡散光のRGBA値
GL_SPECULAR	float型の4次元配列	鏡面光のRGBA値
GL_POSITION	float型の4次元配列	光源の位置(x,y,z,w) 点光源はw=1. (x,y,z)は座標を表す 平行光源はw=0. (x,y,z)は方向ベクトルを表す

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

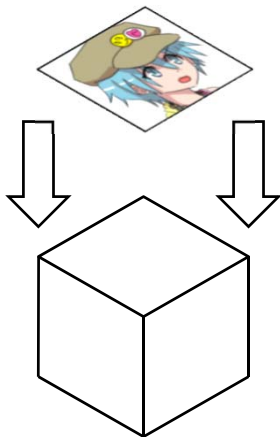
pname	params	パラメータ
GL_AMBIENT	float型の4次元配列	材質の環境RGBA値
GL_DIFFUSE	float型の4次元配列	材質の拡散RGBA値
GL_SPECULAR	float型の4次元配列	材質の鏡面RGBA値
GL_SHININESS	float型のスカラー値	鏡面係数 0以上の値を指定する 小さい数値を指定すると鏡面反射が広がり、表面が粗い材質を表現できる 大きい数値を指定すると鏡面反射がせばまり、表面が滑らかな材質を表現できる

■ 図 4.33——フォンの反射モデルによるハイライトの違い

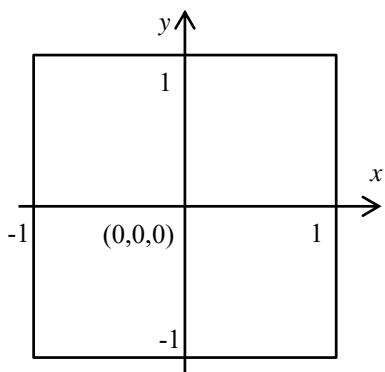
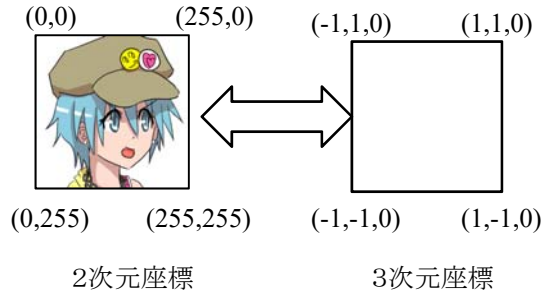


テクスチャマッピングの座標系

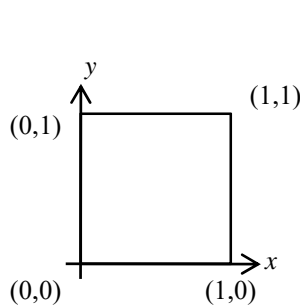
3次元の面に2次元の画像を貼り付ける=テクスチャマッピング



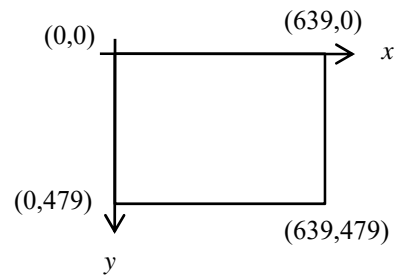
点の座標を対応づける



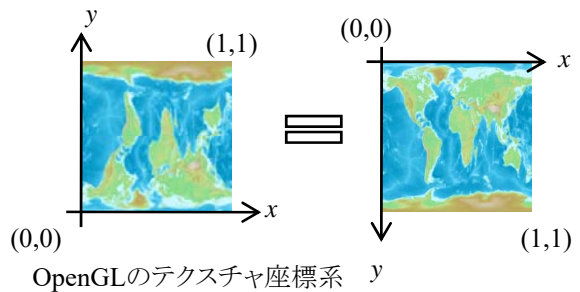
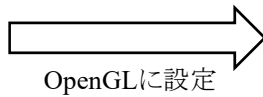
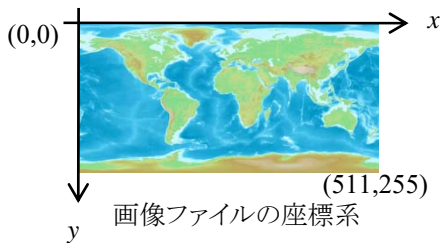
OpenGLの空間の座標系(初期状態)



OpenGLのテクスチャ座標系



画像やディスプレイの座標系



```
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);

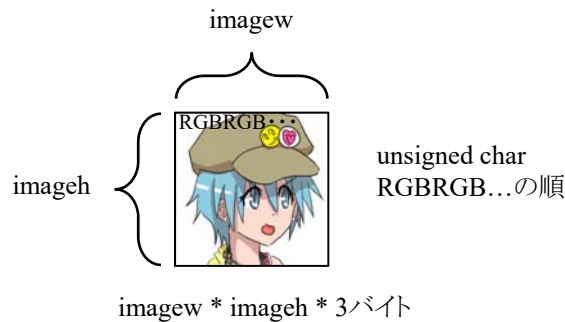
// glClearを呼んだあと、描画したいシーンの命令を実行

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

glutSwapBuffers();
```

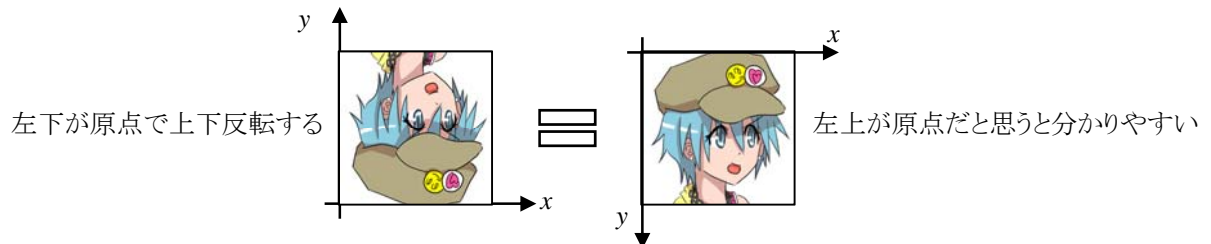
「おまじない」
詳しく知りたい人は教科書等で調べてみよう！

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 64×64 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