OpenGL 入門

新山 祐介 (euske)

1 OpenGL - 概論

1.1 OpenGL の特徴

1. Client-Server Model を採用している。
2. グラフィック描画は手続き的に指示を与える。
3. 計算のバッファリング、コンテキスト、座標系の設定などが可能。

1.2 3D グラフィックスについて

OpenGL はデプスバッファ・アルゴリズム (Depth-Buffer Algorithm あるいは Z-Buffer Algorithm) をもちいて多角形 (polygon ポリゴン) を描画 (レンダリング rendering) する。Client-Server Model であり、X Window System と同じようにクライアント (計算機上の GL ライブラリ) が描画指令を出し、描画はすべて GL 描画サーバ (Indy、O_2 など) がおこなう。描画プロトコルはすべて X の拡張機能 (GLX) として定義されているため、ネットワーク上での扱いは X と同じである。ただし GL ライブラリ自体は X に依存しているわけではないため、実際の描画に先だってウインドウまわりのことを X で実装しておかねばならない。GL ライブラリは X 上のドロワブル (drawable : ウインドウやピクスマップ) に描画することしかできないためである。

OpenGL はポリゴンを素早く描画するため (だけ) に設計されており、デプスバッファもスキャンラインデプスバッファではなくフルスクリーンでの値を保持している。描画は、次のような手順で行なわれる (以下のモデルでは、簡単のためテクスチャマッピング (texture mapping) やアンチエイリアシング (antialiasing) は無視している)。

1. GL サーバに頂点座標、光源情報、材質情報、視野情報が与えられ、最後に描画命令が出される。
2. まず GL サーバは頂点座標から、プリミティブ (primitive) 図形 (おもにポリゴン) を生成する。このさい、高速化のため一度使用した図形を再利用する機能がある (ディスプレイリスト)。また、座標変換は線型写像の積として表現されるが、これは行列のスタックによって変換を蓄積できる。
3. 視覚による変換。これで、視点は原点 (デバイス座標系) に設定される。
4. 射影変換。これは 3D → 2D の変換で、このさい視点を頂点とする四角錘によりプリミティブはクリップ (clip) される。この変換の行列はユーザが設定できるので、視野角の指定も可能。正射影変換も可能となる (ふつうは使わないけど)。
5. ビューポート変換。ウインドウ座標に変換され、描画に必要なバッファ等が初期化される。
6. 描画。デプスバッファを用いてラスタライズしていく。ラスタライズにあたっては、平面だけを描画するフラットシェーディング (flat shading) か、擬似的に曲面を表現するスムースシェーディング (smooth shading) が適用される。なお、OpenGL におけるスムースシェーディングは色を補間するグローシェーディング (glow shading) というアルゴリズムを使用している。

したがって、OpenGL を学ぶのに必要とされるのは基本的な 3D グラフィックスの知識だけである。おそらく以下のような概念の理解を前提とする。

- 線形代数の基礎的知識
- 透視変換とクリッピング
- デプスバッファアルゴリズム
- 拡散光 (diffuse)、反射光 (specular)、環境光 (ambient)

2 実際のプログラミング

今回はとりあえずフォグ (空気遠近法) やフィードバック (画像のイメージ取得) 、曲線の描画等の機能は無視した。また OpenGL には 2D グラフィックスの機能もあるのだが、それらについてはまったく紹介していない。「3D の照光されたオブジェクトの描画」をするための、最低限の関数だけを説明している。

2.1 ライブラリの基本構成

OpenGL の関数群は大きく分けて 3 種類ある。

1. OpenGL コマンド。基本的な描画関数。関数はすべて “gl” の prefix をもつ。
2. OpenGL Utility ルーチン (GLU)。曲線や bitmap 操作に関するもの。関数はすべて “glu” の prefix をもつ。
3. OpenGL X ルーチン (GLX)。X Window System 上での設定や描画の制御。関数はすべて “glx” の prefix をもつ。

ここでは最初の GL 基本コマンドを取りあげる。関数名は、すべて以下の命名形式に従っている。

void **gl** < **FUNCTION** > {#}{**s**|**i**|**f**|**d**}{**v**} (...);

ほとんどの関数は返り値をもたない (あるとしても、主に参照として渡される)。$< FUNCTION >$ の部分にはそれぞれの機能名が入る。あとの #(数字) と s, i, f, d は引数の数と型名である。たとえば **glVertex3f** は次のような形式をとる。

void **glVertex3f** (GLfloat x, GLfloat y, GLfloat z);

また、最後に v のついているものはベクトル形式で、引数として配列を渡すことができるようになっている (正確には配列へのポインタ)。

void **glVertex3fv** (GLfloat v[3]);

このように関数名を見れば引数がわかる。GLfloat というのは GL のために定義された float 型であり、これらはみな $< gl.h >$ の中で定義されている。

関数名での表示	型名	実際はこう
s	GLshort	short (16bit)
i	GLint	int (32bit)
f	GLfloat	float(32bit)
d	GLdouble	double(64bit)

ほかに GLenum という型があり、これはさまざまな定数を指定するために使われる。

2.2 プリミティブの指定

プリミティブ (primitive 基本図形) の形状指定は、形状のタイプをまず指定したあと (glBegin)、頂点命令 (glVertex) をひとつずつ実行していくことにより行う。

- void **glBegin** (GLenum mode [1]);
- void **glEnd** (void);
- void **glFrontFace** (GLenum mode [2]);
- void **glVertex3**{$s|i|f|d$} (GL?? x, GL?? y, GL?? z);
- void **glVertex3**{$s|i|f|d$}**v** (GL?? v[3]);
- void **glNormal3**{$s|i|f|d$} (GL?? x, GL?? y, GL?? z);
- void **glNormal3**{$s|i|f|d$}**v** (GL?? v[3]);

ほんとは glVertex2 や glVertex4 もあるのだが省略。また、照光のためには頂点座標に加えて法線ベクトルを指定しなければならない。そのためには glNormal3* を使う。ふつうこれは glVertex3* の前に指定する。テクスチャマッピングをする際にも、glVertex3* 以前に glTexCoord2* を用いてテクスチャ座標を指定する必要がある (「テクスチャマッピング」の項参照)。いったん法線ベクトルを指定したあとは、glVertex3* で指定される頂点すべてがその法線をもつので、スムースシェーディングを行なわない場合は 1 枚のポリゴンごとに最初に 1 回 glNormal3* を実行すればよい。スムースシェーディングを行なう場合は毎回 glVertex3* を実行する前に glNormal3* をいちいち実行する。

実際の手続きは次のような手順になる。一枚のポリゴンごとに、

1. glBegin(type); で、これから与える頂点がどのような形状かを指定する。おもに使われるのは GL_POLYGON である。このほかに与えられる頂点を 4 つずつに区切って四角形を作る形式などもあるが、ここでは省略。ちなみに、ポリゴンはかならず凸型 (convex) でなければならない。
2. glNormal3*(..); で、ポリゴンの法線を与える (ポリゴンの法線はベクトルの外積によって計算しておく)。
3. 以下、頂点の数だけ…
   (a) テクスチャマッピングを行なう場合は、glTexCoord2*(..); で、頂点のテクスチャ座標を与る。
   (b) glVertex3*(..); で、ポリゴンの頂点座標を与えていく。
4. glEnd(); で、手続き終了。

を実行する。ポリゴンの枚数分だけ、これをくりかえす。glFrontFace はプリミティブのどちらの面が「前面」かを指定する。これに GL_CW を指定すると時計回り、GL_CCW を指定すると反時計回りの頂点指定が「手前の向き」になる。これはプリミティブの材質を指定するときに使われる (「彩色と照光」の項参照)。

[1] mode: GL_POLYGON, GL_QUADS, GL_TRIANGLES ..
[2] mode: GL_CW, GL_CCW

2.3 座標変換とクリッピング

OpenGL でよく使われる 3D 座標系には 2 つの種類がある。ひとつはプリミティブの頂点座標を指定するときにつかう座標系、そしてもうひとつは視点からみた映像を透視変換するときにつかう座標系である。これらはそれぞれ「モデルビュー行列」「射影行列」と呼ばれている。

まずどちらの座標系を変換するかを設定する。つぎに座標変換のコマンドを繰り返していく。それぞれの写像は合成されていくので、座標変換は蓄積されていく。つまり一次変換行列が掛けられていくのである。

- void **glMatrixMode** (GLenum mode [3]);
- void **glLoadIdentify** (void);
- void **glTranslate**{$f|d$} (GL?? x, GL?? y, GL?? z);
- void **glRotate**{$f|d$} (GL?? degree, GL?? x, GL?? y, GL?? z);
- void **glScale**{$f|d$} (GL?? x, GL?? y, GL?? z);
- void **gluLookAt** (GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz);
- void **glFrustum** (GLdouble left, GLdouble righgt, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
- void **glViewport** (GLint x, GLint y, GLsizei width, GLsizei height);
- void **glPushMatrix** (void);
- void **glPopMatrix** (void);

まず glMatrixMode でどの行列について変換するかを設定する。glTranslate*, glRotate*, glScale* はそれぞれ平行移動、回転、拡大縮小の変換である。モデルビュー行列に関しては gluLookAt というユーティリティ関数が使える。これは視点と参照点、そして画面の上を与える点を指定する (レイトレでおなじみ)。

glFrustum は透視変換の際の視野角を決める。具体的には、それぞれの引数に四角錘の頂点を与える。glViewport は最終的な描画領域を決める。glPushMatrix, glPopMatrix で現在の座標系を「行列スタック」に積みこむことができる。これにより、入れ子構造になったモデルを効率よく指定できる。

2.4 彩色と照光

どんなプリミティブも照光されなければ表示されない。OpenGL では、8 つまでの光源を持つことができ、各々は GL_LIGHT0 〜 GL_LIGHT7 の値で指定される。

- void **glLight**{$i|f$} (GLenum light [4] , GLenum pname, GL?? param..);
- void **glLight**{$i|f$}**v** (GLenum light, GLenum pname, GL?? v[ ]);
- void **glMaterial**{$i|f$} (GLenum face [5] , GLenum pname, GL?? param..);
- void **glMaterial**{$i|f$}**v** (GLenum face, GLenum pname, GL?? v[ ]);
- void **glEnable** (GLenum light);

glLight* は 光源とそのパラメータを指定し、その値を設定する。ひとつの光源ごとに以下のようなパラメータがある (一部省略)。

[3] mode: GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE
[4] light: GL_LIGHT0, GL_LIGHT1, ... , GL_LIGHT7
[5] face: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK

パラメータ (pname)	引数	意味
GL_AMBIENT	4	環境光 (R, G, B, A)
GL_DIFFUSE	4	拡散光 (R, G, B, A)
GL_SPECULAR	4	鏡面光 (R, G, B, A)
GL_POSITION	4	光源の位置 (x, y, z, w)
GL_SPOT_DIRECTION	3	スポットの向き (vx, vy, vz)
GL_SPOT_CUTOFF	1	スポットの拡散角 (degree)

GL_POSITION の w 値は $w = 0$ なら平行光源 (無限遠に位置する)、$w \neq 0$ なら点光源あるいはスポットライトとなる。点光源はスポットライトの拡散角が 180 °の特別の場合で、この場合光源は全方向に広がる。平行光源の場合、位置は無視される。そうではない場合、光源の位置はモデルビュー座標系に従う。色の指定は RGBA モデルで行なわれるが、パラメータ A はとりあえず無視できる。

光源を指定したら、それを点灯しなくてはならない。この操作は glEnable で行なう。これ以後、描画するプリミティブにはすべてその光源の影響が表われることになる[6] 。

glMaterial* はこれから描こうとしているプリミティブの材質、つまり「カレント材質」を指定する。プリミティブの前面と背面で異なる材質を設定できる。材質はプリミティブの色も含み、以下のようなパラメータをもつ (一部省略)。

パラメータ (pname)	引数	意味
GL_AMBIENT	4	環境光 (R, G, B, A)
GL_DIFFUSE	4	拡散光 (R, G, B, A)
GL_AMBIENT_AND_DIFFUSE	4	環境光と拡散光 (R, G, B, A)
GL_SPECULAR	4	鏡面反射光 (R, G, B, A)
GL_SHININESS	1	鏡面の指数 (shininess)
GL_EMISSION	4	材質の放射光 (degree)

拡散光はおもに私たちが「物体の色」と呼んでいるものである。鏡面の指数はプリミティブ上でいちばん明るい点 (輝点 hilite) の広がりを示す。放射光は材質みずからが放っている光である。実際に画面上に表われる色は、材質と光源 (あと使用されていればテクスチャ) との相互作用によって決定される。

2.5 テクスチャマッピング

- void **glTexImage2D** (GL_TEXTURE_2D, GLint level, GLint components, GLsizei width, GLsizei height, GLint border, GL_RGB, GL_UNSIGNED_BYTE, const GLvoid* pixels);
- void **glTexParameter**$\{i|f\}$ (GL_TEXTURE_2D, GLenum pname [7] , GL?? param [8]);
- void **glTexCoord2**$\{s|i|f|d\}$ (GL?? s, GL?? t);
- void **glTexCoord2**$\{s|i|f|d\}$v (GL?? v[2]);

テクスチャマッピングを行なうためには、まず OpenGL 上での画像の扱いを知らねばならない。いちばん単純なのは、次のような配列にビットマップデータを格納しておくことである。

GLubyte texImageName[imageWidth][imageHeight][3];

[6] なお、glEnable は光源以外にも OpenGL のさまざまな機能を切り替えるのに使われる。
[7] pname: GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T, GL_TEXTURE_MAG_FILTER, GL_TEXTURE_MIN_FILTER
[8] param: GL_CLAMP, GL_REPEAT, GL_NEAREST, GL_LINER

これで、glTexImage2D に GL_RGB 形式のフォーマットとして渡せる。あとは正確な width, height さえ指定すればよい。ここで配列がピクセル数 × 3 要素できることになるが、[x][y][0] ,[x][y][1], [x][y][2] がそれぞれ Red, Green, Blue の各濃度に対応している。

指定されたテクスチャは (0, 0)-(1, 1) という座標系に変換されて扱われる。これを「テクスチャ座標系」とよび、(s, t) で表わす。glVertex3* で頂点を指定する前に、glTexCoord2* によってその頂点でのテクスチャ座標を指定しておけば、そのとおりテクスチャが貼り付けられる。テクスチャの伸縮にともなう補間や間引きの方法については glTexParameter* で GL_TEXTURE_MAG_FILTER, GL_TEXTURE_MIN_FILTER にパラメータをそれぞれ指定する。GL_NEAREST は負荷が軽いが汚ない。GL_LINER は線型補間をするので滑らかであるが遅い。テクスチャを反復させたいときは glTexParameter* で GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T に GL_REPEAT をそれぞれ指定する。これを指定した以降は、glTexCoord2* によって指定されるテクスチャ座標に上記の範囲外の値を指定することができ、これでテクスチャの反復を指定できる。なお、テクスチャは照光されているため、照明の明るさによって変化して見える。

2.6 ディスプレイリスト

ディスプレイリストは OpenGL の一連の描画コマンド (glBegin, glEnd, glVertex3*, glLight* など) を 2 度以上実行するとき、前に送ったコマンドを再利用することで効率を高めようというものである (コマンドによっては、ディスプレイリストには保存できないものもあるが、ふつうの描画コマンドだけなら大丈夫)。

- void **glNewList** (GLuint list, GLenum mode [9]);
- void **glEndList** (void);
- void **glCallList** (GLuint list);

ディスプレイリストの使用法は、次のとおり。

1. glNewList にこれから登録させるディスプレイリスト番号 (勝手に指定できる) を与えて呼び出す。
2. 一連の描画コマンドを実行する。この際、GL_COMPILE だと送られたコマンドの実行は行なわれない。GL_COMPILE_AND_EXECUTE だと同時に実行もされる。
3. glEndList を実行してリストを完了させる。
4. glCallList でリストを実行する。

なお、glNewList 〜 glEndList は入れ子にできないので注意。

2.7 描画制御・その他

- void **glClear** (GLbitfield mask [10]);
- void **glClearColor** (GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);
- void **glClearDepth** (GLclampd depth);
- void **glShadeModel** (GLenum model [11]);

[9] mode: GL_COMPILE, GL_COMPILE_AND_EXECUTE
[10] mask: GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT
[11] model: GL_FLAT, GL_SMOOTH

- void **glFlush** (void);
- void **glEnable** (GLenum enable);
- void **glDisable** (GLenum enable);

glClear はマスクビットで指定されたバッファをクリアする。クリアするときの値はカラーバッファ、デプスバッファそれぞれ glClearColor, glClearDepth によって指定できる。glShadeModel はフラットシェーディングかスムースシェーディングかを指定する。glFlush はバッファリングされている全 GL リクエストをフラッシュし描画を実行する。glEnable, glDisable は前にもちょっと出てきたが、OpenGL の諸機能をオン/オフするのに使われる。

OpenGL の典型的な初期設定 :

```
glEnable(GL_LIGHTING);                                  // 照光する
glEnable(GL_NORMALIZE);                                 // 法線ベクトルをつねに正規化
glEnable(GL_TEXTURE_2D);                                // テクスチャを使用
glEnable(GL_DEPTH_TEST);                                // Depth-Buffer を使用
glTexEnv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE); // テクスチャを変調
glLightModelf(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE); // 両面に照光
glShadeModel(GL_FLAT);                                  // フラットシェーディング
glClearColor(0.0, 0.0, 0.0, 1.0);                       // 黒で消去
glClearDepth(0.0);                                      // デプスを 0 で消去
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

3 補足 - GLX の使用法

OpenGL はハードウエアに依存しないライブラリなので、実装のためには若干 X の知識を必要とする。

- XVisualInfo* **glXChooseVisual** (Display* dpy, int screen, int* attriblist);
- int **glXGetConfig** (Display* dpy, XVisualInfo* vis, int attrib, int* value);
- GLXContext **glXCreateComtext** (Display* dpy, XVisualInfo* vis, GLXContext shareList, Bool direct);
- Bool **glXMakeCurrent** (Display* dpy, GLXDrawable draw, GLXContext ctx);
- GLXPixmap **glXCreateGLXPixmap** (Display* dpy, XVisualInfo* vis, Pixmap pixmap);
- void **glXSwapBuffers** (Display* dpy, Window window);

GLX の関数を使った、おおまかな実装は次のようになる。

1. まず glXChooseVisual で 適切な Visual を選択する。
2. glXCreateContext でレンダリングコンテキストを作る。
3. カラーマップを用意し、自分の使いたいカラーをストアする。
4. X ウインドウを作成、マップする。
5. Expose イベントで OpenGL のコマンドを使い描画、あとはふつうのイベント処理。

glXChooseVisual には、まずグラフィックに必要な能力を要求する。GL サーバはこれによって必要はビジュアルを選定する。あとはこれを glXCreateContext に渡して、適当なドロワブルを作ってからそれを glXMakeCurrent でコンテキストと結びつければ OK である。アニメーションを行ないたいときは、一回描画するごとに glXSwapBuffers を実行すれば、自動的にバッファを切りかえてくれるらしい。ただし、この場合は最初に GLX_DOUBLEBUFFER 属性を要求しておくこと。

GLX の典型的な初期設定 :

```
int list[] = {GLX_RGBA,           // RGB モード指定
              GLX_RED_SIZE, 1,    // Red, Green, Blue の各バイトサイズ
              GLX_GREEN_SIZE, 1,
              GLX_BLUE_SIZE, 1,
              GLX_DEPTH_SIZE, 1,
              // GLX_DOUBLEBUFFER, // アニメーションを行う場合は指定
              None};              // リストの終了
XVisualInfo* myvisual;
GLXContext mycontext;
Display* dpy;
Window win;
Colormap colmap;

dpy = XOpenDisplay(NULL);         // X サーバとの接続
myvisual = glXChooseVisual(dpy, 0, list); // ビジュアルの選定
mycontext = glXCreateContext(dpy, myvisual, None, GL_FALSE); // X用コンテキスト作成
colmap = XCreateColormap(...);    // カラーマップを作る
win = XCreateSimpleWindow(...);   // トップレベルウインドウを作る
XSetWMProperties(...);            // ウインドウマネージャの設定
XMapWindow(...);                  // ウインドウをマップする
glXMakeCurrent(dpy, win, mycontext); // コンテキストとウインドウを結びつける

/*
  イベント処理...
*/
```