|  |
| --- |
| 博士課程教育リーディングプログラム  実世界データ循環学リーダー人材育成プログラム |
| データツールファースト |
| OpenGL入門 |

|  |
| --- |
| 名古屋大学  2014/03/02 |

* 記載されている会社名、製品名等は、各社の登録商標または商標です。

目次

[1. はじめに 1](#_Toc381575109)

[1.1 OpenGLとは 1](#_Toc381575110)

[1.2 構成 1](#_Toc381575111)

[1.3 インストールおよびコンパイル 2](#_Toc381575112)

[1.4 OpenGLの関数名 2](#_Toc381575113)

[1.5 色の指定 3](#_Toc381575114)

[1.6 例:ウィンドウの生成 3](#_Toc381575115)

[2. 図形の描画 6](#_Toc381575116)

[2.1 図形の描画 6](#_Toc381575117)

[2.2 投影法 7](#_Toc381575118)

[2.3 例:2次元図形の描画 8](#_Toc381575119)

[[演習] 10](#_Toc381575120)

[2.4 幾何変換 10](#_Toc381575121)

[2.5 例:3次元図形の描画 11](#_Toc381575122)

[[演習] 12](#_Toc381575123)

[2.6 隠面処理 13](#_Toc381575124)

[2.7 例:隠面処理 13](#_Toc381575125)

[[演習] 15](#_Toc381575126)

[3. イベント処理 16](#_Toc381575127)

[3.1 キー 16](#_Toc381575128)

[3.2 例:キーイベント処理 17](#_Toc381575129)

[[演習] 19](#_Toc381575130)

[3.3 マウス 19](#_Toc381575131)

[3.4 ウィンドウのリシェープ 20](#_Toc381575132)

[4. アニメーション 22](#_Toc381575133)

[4.1 アニメーションの設定 22](#_Toc381575134)

[4.2 ダブルバッファ 22](#_Toc381575135)

[4.3 例:アニメーション 22](#_Toc381575136)

[[演習] 25](#_Toc381575137)

[5. シェーディング 26](#_Toc381575138)

[5.1 シェーディングの設定 26](#_Toc381575139)

[5.2 表面属性 27](#_Toc381575140)

[5.3 スムースシェーディング 28](#_Toc381575141)

[5.4 例:シェーディング 28](#_Toc381575142)

[[演習] 31](#_Toc381575143)

[6. 光源設定 32](#_Toc381575144)

[6.1 光源の種類と設定 32](#_Toc381575145)

[6.2 例:光源設定 33](#_Toc381575146)

[[演習] 37](#_Toc381575147)

[7. テクスチャマッピング 38](#_Toc381575148)

[7.1 テクスチャマッピングの設定 38](#_Toc381575149)

[7.2 例:テクスチャマッピング 40](#_Toc381575150)

[[演習] 44](#_Toc381575151)

[7.3 自動マッピング 45](#_Toc381575152)

[7.4 例:自動マッピング 46](#_Toc381575153)

[[演習] 50](#_Toc381575154)

[8. 参考文献 52](#_Toc381575155)

[9. 変更履歴 53](#_Toc381575156)

# はじめに

## OpenGLとは

「OpenGL」は、3次元CGのグラフィックスライブラリです。当初は「Silicon Graphics International(SGI)」社により開発されたものですが、「OpenGL Architecture Review Board(ARB)」を経て、現在は「Khronos」という非営利団体が仕様を策定しています。

Windows、Mac OS、UNIX系など多数のプラットフォームに対応していること、ハードウェアの機能を利用して高速に動作すること、3次元CGを比較的簡単に作成できることから、幅広く使われています。主に、シミュレーション解析、CAD、ゲームやマルチメディア用途などで利用されています。

また、携帯電話や家電など組み込み用途向けのサブセット「OpenGL ES」もあります。OpenGL ESも、iOS、Android、Windows Mobileなど様々な組み込み向けOSに対応しています。

本書は、OpenGLの概要と、基本的な使い方を習得することを目的とした入門書です。OpenGLのAPIは多数あるため、本書では、そのすべてを説明することを目的としません。OpenGLをもっと詳しく学ぶには、8章の参考文献を利用してください。

また、OpenGLはプラットフォームだけでなく言語にも依存しませんが、本書ではC言語を使ってOpenGLの機能を説明します。

## 構成

OpenGLは、以下の3つのライブラリから構成されています。

* OpenGLライブラリ(GL)

点、線、ポリゴンの描画、色やテクスチャの制御、幾何変換や投影変換、シェーディングなどの基本的な処理を行うためのライブラリです。

ヘッダ: GL/gl.h

ライブラリ: libGL.so

* OpenGLユーティリティライブラリ(GLU)

OpenGLの上位ライブラリであり、OpenGLのコマンドを組み合わせた複雑な処理を行います。

ヘッダ: GL/glu.h

ライブラリ: libGLU.so

* glutライブラリ(GLUT)

ウィンドウや入力イベントの管理など、プラットフォームに依存する処理を行うためのライブラリです。

ヘッダ: GL/glut.h

ライブラリ: libglut.so

## インストールおよびコンパイル

CentOSなどでは、以下の手順でパッケージをインストールします。

|  |
| --- |
| # yum install mesa-libGL-devel mesa-libGLU-devel freeglut-devel |

たとえば、foo.cをコンパイルするには、以下のように実行します。

|  |
| --- |
| $ gcc -O -o foo foo.c -lglut -lGLU -lGL |

実際に使用するライブラリに合わせて、リンカオプションを変更してください。

Windowsの場合は、Visual Studioなどに含まれています。グラフィックスボードメーカーが提供するSDKもあります。前者で、コマンドラインからコンパイルするには、以下のように実行します。

|  |
| --- |
| $ cl -o foo foo.c -lglut -lGLU -lGL |

## OpenGLの関数名

OpenGLの関数名は、接頭語、コマンド名および接尾語から構成されてます。

gl Vertex 3f (x, y, z);

接頭語 コマンド名 接尾語 引数

接頭語は、ライブラリを表すためのものです。「gl」の場合はGL、「glu」の場合はGLU、「glut」の場合はGLUTの関数であることがわかります。

コマンド名は、処理する内容を表すためのものです。

接尾語は、引数の数や型を表すためのものです。数字は引数の数、アルファベットは引数の型を表します。たとえば、「f」は「float」、「d」は「double」、「i」は「int」などです。ただし、GLの描画関数以外では、接尾語がつかないものが多くあります。

## 色の指定

OpenGLでは、色の指定は、光の3原色であるR(Red:赤)、G(Green:緑)、B(Blue:青)と、A(Alpha:透明度)による「RGBA」モードと、インデックスで指定する「INDEX」モードがあります。本書では、前者のRGBAモードを使用します。

色を指定する関数には、描画時の色の指定を行うglColor3f()や、クリア時の色の設定を行うglClearColor()などがあります。いずれも、引数にRGBAの各値を指定します。

## 例:ウィンドウの生成

まずは、ウィンドウを生成する簡単なプログラムを例に、OpenGLを使ったプログラムの基本構成を説明します。

ウィンドウを生成するプログラム

|  |
| --- |
| #include <GL/gl.h>  #include <GL/glut.h>  static void display\_callback(void)  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glFlush();  }  static void keyboard\_callback(unsigned char key, int x, int y)  {  if (key == '\e') {  exit(0);  }  }  int main(int argc, char \*argv[])  {  glutInit(&argc, argv);  glutInitWindowPosition(0, 0);  glutInitWindowSize(320, 240);  glutInitDisplayMode(GLUT\_RGBA);  glutCreateWindow(argv[0]);  glClearColor(0.0, 0.0, 0.0, 1.0);  glutKeyboardFunc(keyboard\_callback);  glutDisplayFunc(display\_callback);  glutMainLoop();  return 0;  } |

main()関数では、以下の処理を行っています。

複雑なプログラムでは、他の初期化処理がさらに必要ですが、基本的には以下のことを行います。

* glutInit()関数による初期化
* glutInitWindowPosition()およびglutInitWindowSize()関数で、生成するウィンドウの位置と大きさを指定
* glutInitDisplayMode()関数で、ディスプレイモードを「RGBA」に設定
* glutCreateWindow()関数で、タイトルがプログラム名(argv[0])のウィンドウを生成
* glInitColor()関数で、ウィンドウ内の色を黒に指定
* glutKeyboardFunc()関数で、キーイベント発生時に呼び出されるコールバック関数を指定(詳細は3.1節で説明します)
* glutDisplayFunc()関数で、再描画を行う関数を指定
* glutMainLoop()関数で、ループ処理に移行(イベント待ち状態)

glutKeyboardFunc()関数で指定したコールバック関数keyboard\_callback()では、[Esc]キーが押されたとき終了(exit)するよう実装されています。

また、glutDidplayFunc()関数で指定したコールバック関数display\_callback()では、以下の処理を行っています。

* glClear()関数で、カラーバッファ(GL\_COLOR\_BUFFER\_BIT)をクリア
* glFlush()関数で、発行したOpenGLのコマンドをすべて実行

これをコンパイルして実行すると、図 1のウィンドウが表示されます。

[Esc]キーを押すと、プログラムが終了します。



図 1ウィンドウ生成プログラムの実行例

# 図形の描画

## 図形の描画

点、線もしくはポリゴンを描画するには、以下のように記述します。

|  |
| --- |
| glBegin(描画モード);  glVertex3f(x0, y0, z0);  glVertex3f(x1, y1, z1);  …  glEnd(描画モード); |

glBegin()関数とglEnd()関数の間に、点を描画するglVertex3f()関数などを呼び出します。

glVertex3f()の引数は、点の座標です。

|  |
| --- |
| glVertex3f(x, y, z); |

主な描画モードを、以下に示します。

|  |  |
| --- | --- |
| **描画モード名** | **概要** |
| GL\_POINTS | 点の描画 |
| GL\_LINES | 線の描画 |
| GL\_LINE\_STRIP | 連結された線の描画(両端は開放) |
| GL\_LINE\_LOOP | 連結された線の描画(両端も連結) |
| GL\_POLYGON | 多角形の描画 |
| GL\_TRIANGLE\_FAN | 始点を中心とする、連結された三角形ポリゴンの描画 |
| GL\_TRIANGLE\_STRIP | 連結された三角形ポリゴンの描画 |
| GL\_QUAD\_STRIP | 連結された四角形ポリゴンの描画 |

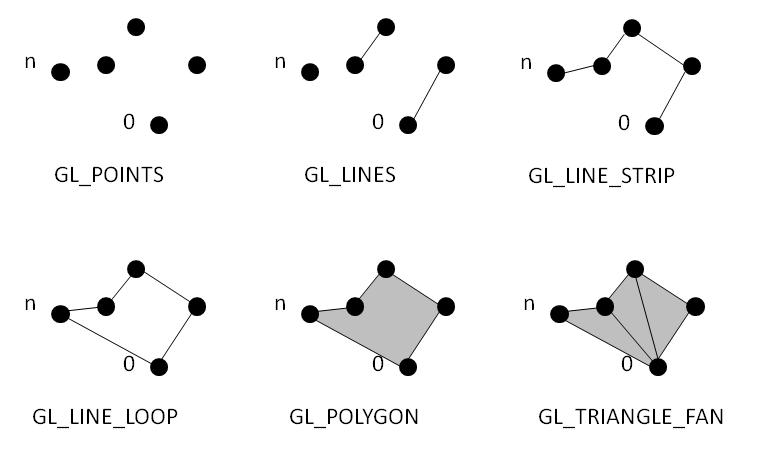


図 2主な描画モード

ポリゴンには表裏があります。反時計回りに描画した側が表になります。

## 投影法

コンピュータ内部の3次元座標系を「ワールド座標系」、ディスプレイ上の2次元平面の座標系を「スクリーン座標系」と呼びます。ワールド座標系にある3次元の物体を、2次元のスクリーン座標系に映すことを「投影法」と言います。投影法には、「平行投影」と「透視投影」が主に用いられます(図 3)。

平行投影では、平行に投影を行うため、物体の寸法を正確に表せます。ただし、遠くの物体も近くの物体も同じ寸法になるため、遠近感を表すことができません。

透視投影では、視点を中心に投影を行うため、現実に近い遠近感を表すことができます。

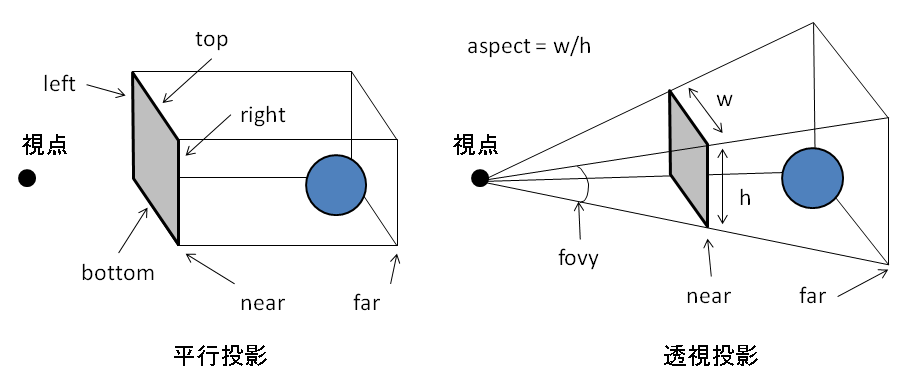


図 3平行投影と透視投影

平行投影を行うには、glOrtho()関数を用います。

引数は、投影面の左右(x座標:left,right)、下上(y座標:bottom,top)および前後(z座標:near,far)です。

|  |
| --- |
| glOrtho(left, right, bottom, top, near, far); |

透視投影を行うには、gluPerspective()関数を用います。

引数は、上下の開き角(fovy)、投影面の幅・高さの比(aspect)および前後(near,far)です。

|  |
| --- |
| gluPerspective(fovy, aspect, near, far); |

## 例:2次元図形の描画

平行投影による2次元図形の描画の例を示します。

三角形を描画するプログラム

|  |
| --- |
| #include <GL/gl.h>  #include <GL/glut.h>  static void display\_callback(void)  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glBegin(GL\_POLYGON);  glColor3f(1.0, 0.0, 0.0);  glVertex3f(0.0, -0.8, 0.0);  glColor3f(0.0, 1.0, 0.0);  glVertex3f(-0.8, 0.8, 0.0);  glColor3f(0.0, 0.0, 1.0);  glVertex3f(0.8, 0.8, 0.0);  glEnd();  glFlush();  }  static void keyboard\_callback(unsigned char key, int x, int y)  {  if (key == '\e') {  exit(0);  }  }  int main(int argc, char \*argv[])  {  glutInit(&argc, argv);  glutInitWindowPosition(0, 0);  glutInitWindowSize(320, 240);  glutInitDisplayMode(GLUT\_RGBA);  glutCreateWindow(argv[0]);  glClearColor(0.0, 0.0, 0.0, 1.0);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  glOrtho(-1.0, 1.0, -1.0, 1.0, 0.0, 0.0);  glutKeyboardFunc(keyboard\_callback);  glutDisplayFunc(display\_callback);  glutMainLoop();  return 0;  } |

1.5節のプログラムと異なる個所(赤色の部分)のみ説明します。

main()関数では、平行投影の設定を行うため、glMatrixMode(GL\_PROJECTION)およびglLoadIdentity()で初期化を行います。その後、glOrtho()関数で平行投影の設定を行います。

描画のコールバック関数display\_callback()では、三角形ポリゴンを1つ描画しています。各点ごとに、glColor3f()関数で色を指定しています。引数は、RGBAの各値です。

これをコンパイルして実行すると、図 4のウィンドウが表示されます。

[Esc]キーを押すと、プログラムが終了します。

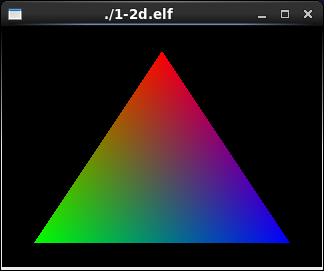


図 42次元図形描画プログラムの実行例

### [演習]

四角形や多角形を描画するプログラムを作成しましょう。

色も変えてみましょう。

## 幾何変換

「幾何変換」により、ワールド座標系の位置の変更、回転、拡大・縮小の操作が可能です。

平行移動するには、glTranslatef()関数を用います。

引数は、各軸方向の移動量(tx, ty, tz)です。

|  |
| --- |
| glTranslatef(tx, ty, tz); |

回転するには、glRotatef()関数を用います。

引数は、回転角(angle)と、回転軸のベクトル(x, y, z)です。

|  |
| --- |
| glRotatef(angle, x, y, z); |

拡大・縮小を行うには、glScalef()関数を用います。

引数は、各軸方向の拡大率(sx, sy, sz)です。マイナスの場合は反転します。

|  |
| --- |
| glScalef(sx, sy, sz); |

## 例:3次元図形の描画

透視投影による3次元図形の描画の例を示します。

立方体を描画するプログラム

|  |
| --- |
| #include <GL/gl.h>  #include <GL/glu.h>  #include <GL/glut.h>  static void display\_callback(void)  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f(1.0, 1.0, 1.0);  glutWireCube(1.0);  glFlush();  }  static void keyboard\_callback(unsigned char key, int x, int y)  {  if (key == '\e') {  exit(0);  }  }  int main(int argc, char \*argv[])  {  glutInit(&argc, argv);  glutInitWindowPosition(0, 0);  glutInitWindowSize(320, 320);  glutInitDisplayMode(GLUT\_RGBA);  glutCreateWindow(argv[0]);  glClearColor(0.0, 0.0, 0.0, 1.0);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(45.0, 1.0, 2.0, 10.0);  glTranslatef(0.0, 0.0, -4.0);  glRotatef(30.0, 1.0, 1.0, 0.0);  glutKeyboardFunc(keyboard\_callback);  glutDisplayFunc(display\_callback);  glutMainLoop();  return 0;  } |

main()関数では、gluPerspective()関数で透視投影の設定を行った後、glTranslatef()およびglRotatef()で幾何変換を行っています。

描画のコールバック関数display\_callback()では、glutWireCube()関数を用いて、立方体を描画しています。引数は辺の長さです。

これをコンパイルして実行すると、図 5のウィンドウが表示されます。

[Esc]キーを押すと、プログラムが終了します。

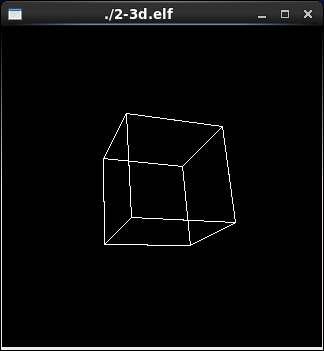


図 53次元図形描画プログラムの実行例

### [演習]

幾何変換をいくつか使用して、表示がどのように変わるか確認しましょう。

## 隠面処理

複数の物体をそのまま描画すると、遠近に関係なく、後から描画した物体が手前に表示されます。OpenGLでは、「zバッファ法」により、視点から物体までの距離(z値)が最も近い可視面だけを描画する「隠面処理」を行うことが可能です。

隠面処理の設定方法は、以下の通りです。

* glutInitDisplayMode()関数の引数に「GLUT\_DEPTH」を指定して、zバッファの利用を宣言
* 描画前に、glClear()関数の引数に「GL\_DEPTH\_BUFFER\_BIT」を指定してzバッファを初期化
* 隠面処理の実施範囲をglEnable(GL\_DEPTH\_TEST;～glDisable(GL\_DEPTH\_TEST)で囲む

## 例:隠面処理

隠面処理を行うプログラムの例を示します。

隠面処理を行うプログラム

|  |
| --- |
| #include <GL/gl.h>  #include <GL/glu.h>  #include <GL/glut.h>  static void display\_callback(void)  {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glEnable(GL\_DEPTH\_TEST);  glColor3f(0.0, 1.0, 1.0);  glutSolidCube(1.0);  glColor3f(1.0, 0.0, 0.0);  glRectf(-1.5, -0.5, 1.5, 0.5);  glDisable(GL\_DEPTH\_TEST);  glFlush();  }  static void keyboard\_callback(unsigned char key, int x, int y)  {  if (key == '\e') {  exit(0);  }  }  int main(int argc, char \*argv[])  {  glutInit(&argc, argv);  glutInitWindowPosition(0, 0);  glutInitWindowSize(320, 320);  glutInitDisplayMode(GLUT\_RGBA | GLUT\_DEPTH);  glutCreateWindow(argv[0]);  glClearColor(0.0, 0.0, 0.0, 1.0);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(45.0, 1.0, 2.0, 10.0);  glTranslatef(0.0, 0.0, -4.0);  glRotatef(30.0, 1.0, 1.0, 0.0);  glutKeyboardFunc(keyboard\_callback);  glutDisplayFunc(display\_callback);  glutMainLoop();  return 0;  } |

main()関数のglutInitDisplayMode()で、GLUT\_DEPTHを指定して、zバッファの利用を宣言しています。

描画のコールバック関数display\_callback()では、glClear()でGL\_DEPTH\_BUFFER\_BITを指定してzバッファを初期化し、立方体と四角形の描画を、glEnable(GL\_DEPTH\_TEST)とglDisable(GL\_DEPTH\_TEST)で囲っています。

これをコンパイルして実行すると、図 6のウィンドウが表示されます。

[Esc]キーを押すと、プログラムが終了します。

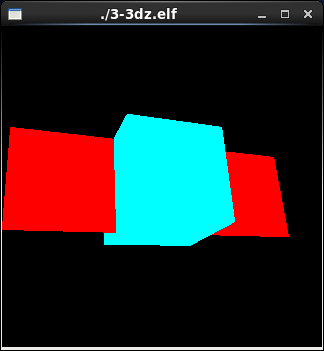


図 6隠面処理プログラムの実行例

### [演習]

隠面処理を行わないとどのように表示されるか、確認しましょう。

# イベント処理

## キー

OpenGLでは、入力されたキーに応じた処理を行うことができます。

一般キーが押された時のコールバック関数の登録は、glutKeyboardFunc()関数で行います。

|  |
| --- |
| glutKeyboardFunc(callback); |

引数はコールバック関数です。

|  |
| --- |
| void callback(unsigned char key, int x, int y); |

keyには押されたキーのASCIIコード、xおよびyにはマウスポインタの位置が渡ります。

特殊キーが押された時のコールバック関数の登録は、glutSpecialFunc()関数で行います。

|  |
| --- |
| glutSpecialFunc(callback); |

引数はコールバック関数です。

|  |
| --- |
| void callback(int key, int x, int y); |

keyには押された特殊キーの値、xおよびyにはマウスポインタの位置が渡ります。

主な特殊キーの値と意味を以下に示します。

|  |  |
| --- | --- |
| **特殊キーの値** | **意味** |
| GLUT\_KEY\_LEFT,RIGHT,UP,DOWN | カーソルキー |
| GLUT\_KEY\_F1～F12 | ファンクションキー |
| GLUT\_KEY\_PAGE\_UP,DOWN | PageUp, PageDownキー |
| GLUT\_KEY\_HOME,END | Home, Endキー |
| GLUT\_KEY\_INSERT | Insertキー |

## 例:キーイベント処理

キーイベント処理を行うプログラムの例を示します。

隠面処理を行うプログラム

|  |
| --- |
| #include <GL/gl.h>  #include <GL/glu.h>  #include <GL/glut.h>  static void display\_callback(void)  {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glEnable(GL\_DEPTH\_TEST);  glColor3f(0.0, 1.0, 1.0);  glutSolidCube(1.0);  glColor3f(1.0, 0.0, 0.0);  glRectf(-1.5, -0.5, 1.5, 0.5);  glDisable(GL\_DEPTH\_TEST);  glFlush();  }  static void special\_callback(int key, int x, int y)  {  switch (key) {  case GLUT\_KEY\_LEFT:  glRotatef(-10, 0.0, 1.0, 0.0);  break;  case GLUT\_KEY\_RIGHT:  glRotatef(10, 0.0, 1.0, 0.0);  break;  case GLUT\_KEY\_UP:  glRotatef(10, 1.0, 0.0, 0.0);  break;  case GLUT\_KEY\_DOWN:  glRotatef(-10, 1.0, 0.0, 0.0);  break;  case GLUT\_KEY\_PAGE\_UP:  glTranslatef(0.0, 0.0, 0.2);  break;  case GLUT\_KEY\_PAGE\_DOWN:  glTranslatef(0.0, 0.0, -0.2);  break;  }  glutPostRedisplay();  }  static void keyboard\_callback(unsigned char key, int x, int y)  {  if (key == '\e') {  exit(0);  }  }  int main(int argc, char \*argv[])  {  glutInit(&argc, argv);  glutInitWindowPosition(0, 0);  glutInitWindowSize(320, 320);  glutInitDisplayMode(GLUT\_RGBA | GLUT\_DEPTH);  glutCreateWindow(argv[0]);  glClearColor(0.0, 0.0, 0.0, 1.0);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(45.0, 1.0, 2.0, 10.0);  glTranslatef(0.0, 0.0, -4.0);  glutKeyboardFunc(keyboard\_callback);  glutSpecialFunc(special\_callback);  glutDisplayFunc(display\_callback);  glutMainLoop();  return 0;  } |

一般キーに関しては、今までの例と同じです。コールバック関数keyboard\_callback()で、押されたキーが[Escape]の場合、exit()によりプログラムを終了します。

main()関数では、glutSpecialFunc()関数で、特殊キーが押された時のコールバック関数を登録しています。コールバック関数special\_callback()では、カーソルキーで回転、[PageUp][PageDown]キーでz方向の移動を行っています。

また、最後にglutPostRedisplay()関数を呼び出して、ディスプレイのコールバック関数を実行しています。

これをコンパイルして実行すると、図 7のウィンドウが表示されます。

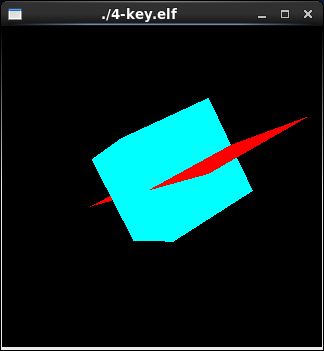


図 7キーイベント処理プログラムの例

### [演習]

[+][-]が押された時、拡大・縮小を行うようにしてみましょう。

## マウス

マウス操作に関するイベント処理には、ボタンのクリックとドラッグがあります。

マウスボタンをクリックした時のコールバック関数の登録は、glutMouseFunc()関数で行います。

|  |
| --- |
| glutMouseFunc(callback); |

引数はコールバック関数です。

|  |
| --- |
| void callback(int button, int state, int x, int y); |

buttonには押されたボタンの値、stateには状態(GLUT\_DOWNかGLUT\_UP)、xおよびyにはマウスポインタの位置が渡ります。

ボタンの値と意味を以下に示します。

|  |  |
| --- | --- |
| **ボタンの値** | **意味** |
| GLUT\_LEFT\_BUTTON | 左ボタン |
| GLUT\_MIDDLE\_BUTTON | 中ボタン |
| GLUT\_RIGHT\_BUTTON | 右ボタン |

マウスボタンをドラッグした時のコールバック関数の登録は、glutMotionFunc()関数で行います。

|  |
| --- |
| glutMotionFunc(callback); |

引数はコールバック関数です。

|  |
| --- |
| void callback(int x, int y); |

xおよびyにはマウスポインタの位置が渡ります。

## ウィンドウのリシェープ

OpenGLでは、ウィンドウのリシェープ(サイズ変更)のイベント処理も行えます。

リシェープ時のコールバック関数の登録は、glutReshapeFunc()関数で行います。

|  |
| --- |
| glutReshapeFunc(callback); |

引数はコールバック関数です。

|  |
| --- |
| void callback(int width, int height); |

widthおよびheightにはウィンドウサイズが渡ります。

# アニメーション

## アニメーションの設定

OpenGLでは、イベント処理を行っていないアイドル状態のときに、アニメーション処理を行うことができます。アイドル時のコールバック関数の登録は、glutIdleFunc()関数で行います。

|  |
| --- |
| glutIdleFunc(callback); |

引数はコールバック関数です。

|  |
| --- |
| void callback(void); |

一般的には、アイドルコールバック関数内で、描画に関連するパラメータ等を操作し、glutPostRedisplay()関数で再描画を行います。

## ダブルバッファ

表示しているフレームバッファに直接描画していると、途中の画面が表示されるため、ちらつきが生じます。これを解消するには、「ダブルバッファ」を用います。

ダブルバッファの設定方法は、以下の通りです。

* glutInitDisplayMode()関数の引数に「GLUT\_DOUBLE」を指定して、ダブルバッファの利用を宣言
* 描画後に、glutSwapBuffers()関数を呼び出してバッファをスワップ

## 例:アニメーション

アニメーションを行うプログラムの例を示します。

アニメーションを行うプログラム

|  |
| --- |
| #include <GL/gl.h>  #include <GL/glu.h>  #include <GL/glut.h>  static void display\_callback(void)  {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glEnable(GL\_DEPTH\_TEST);  glColor3f(0.0, 1.0, 1.0);  glutSolidCube(1.0);  glColor3f(1.0, 0.0, 0.0);  glRectf(-1.5, -0.5, 1.5, 0.5);  glDisable(GL\_DEPTH\_TEST);  glutSwapBuffers();  glFlush();  }  static void keyboard\_callback(unsigned char key, int x, int y)  {  if (key == '\e') {  exit(0);  }  }  static void idle\_callback(void)  {  glRotatef(0.1, 1.0, 0.5, 0.2);  glutPostRedisplay();  }  int main(int argc, char \*argv[])  {  glutInit(&argc, argv);  glutInitWindowPosition(0, 0);  glutInitWindowSize(320, 320);  glutInitDisplayMode(GLUT\_RGBA | GLUT\_DEPTH | GLUT\_DOUBLE);  glutCreateWindow(argv[0]);  glClearColor(0.0, 0.0, 0.0, 1.0);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(45.0, 1.0, 2.0, 10.0);  glTranslatef(0.0, 0.0, -4.0);  glRotatef(30.0, 1.0, 1.0, 0.0);  glutKeyboardFunc(keyboard\_callback);  glutDisplayFunc(display\_callback);  glutIdleFunc(idle\_callback);  glutMainLoop();  return 0;  } |

main()関数では、glutInitDisplayMode()関数にGLUT\_DOUBLEを指定して、ダブルバッファの利用を宣言しています。また、glutIdleFunc()関数で、アイドル時のコールバック関数を登録しています。コールバック関数idle\_callback()では、glRotatef()関数による回転処理を行った後、glutPostRedisplay()で再描画を行います。

ディスプレイのコールバック関数display\_callback()では、描画完了後にglutSwapBuffers()関数でバッファのスワップを行っています。

これをコンパイルして実行すると、図 8のウィンドウが表示されます。

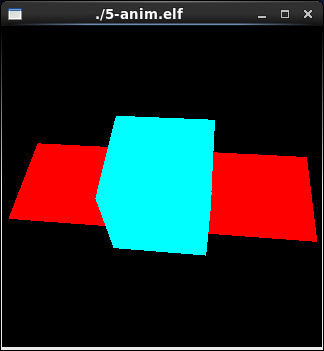
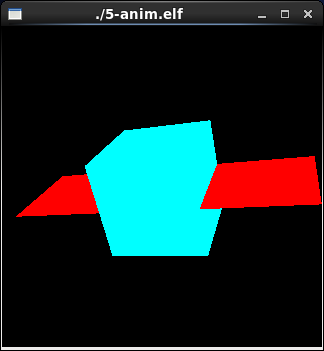


図 8アニメーションプログラムの実行例

### [演習]

回転以外の処理を行ってみましょう。

また、マウスイベントなどの処理も追加してみましょう。

# シェーディング

## シェーディングの設定

「シェーディング」とは、物体に陰影をつける処理のことです。物体表面で反射する光の成分は、図 9に示す3つの成分からなります。

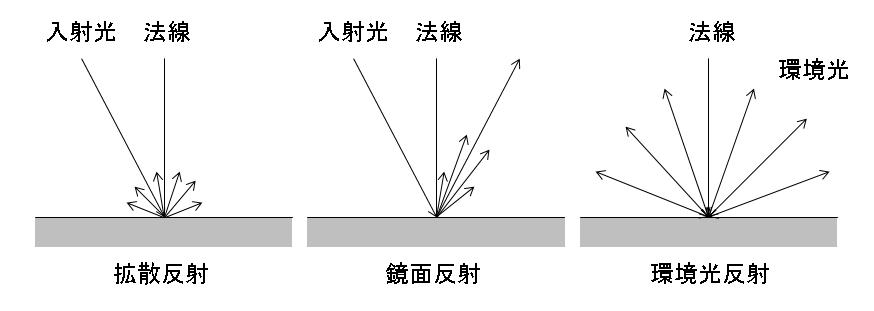


図 9物体表面の反射成分

* 拡散反射成分(DIFFUSE)

ざらついた光沢のない物体表面で起こる光の反射の成分。すべての方向へ一様に散乱するため、視点の方向には依存せず、入射角と、強度および表面の向き(法線ベクトル)から計算される。主に物体の色を反映する。

* 鏡面反射成分(SPECULAR)

物体表面が鏡や金属のように強く反射する成分。ハイライトと呼ばれる輝点を生ずる。

* 環境光反射成分(AMBIENT)

すべての方向から弱く照らされる間接光の成分。これがないと影が真っ黒になる。

シェーディングの設定方法は、以下の通りです。

* glEnable(GL\_LIGHT0)で光源を有効化

GL\_LIGHT0は、デフォルトの光源であり、z軸のマイナス方向を照らす白色の平行光源である。また、光源は、GL\_LIGHT0～7の8個を同時利用可能。

* glNormal3f(x, y, z)でポリゴンに対する法線ベクトルを指定

x,y,zは法線ベクトルの成分で、長さが1の単位ベクトルを指定する必要がある。あるいは、glEnable(GL\_NORMALIZE)～glDisable(GL\_NORMALIZE)で囲む。

* シェーディング対象処理をglEnable(GL\_LIGHTING)～glDisable(GL\_LIGHTING)で囲む

## 表面属性

「表面属性」とは、物体表面が持つ光の反射特性のことです。前述の3つの反射成分から、物体の色や質感を表現できます。

拡散反射成分を設定するには、以下を行います。

|  |
| --- |
| float values[] = { R, G, B, A };  glMaterialfv(face, GL\_DIFFUSE, values); |

faceには、設定する面を以下のいずれかで指定します。valuesは設定地で、各値の範囲は、0.0～1.0です。

|  |  |
| --- | --- |
| **設定値** | **対象** |
| GL\_FRONT | ポリゴンの表面のみに設定 |
| GL\_BACK | ポリゴンの裏面のみに設定 |
| GL\_FRONT\_AND\_BACK | ポリゴンの表裏ともに設定 |

鏡面反射成分を設定するには、以下を行います。

|  |
| --- |
| float values[] = { R, G, B, A };  glMaterialfv(face, GL\_SPECULAR, values); |

引数はいずれも前述の通りです。

また、鏡面反射光の鋭さを、以下で設定できます。

|  |
| --- |
| glMaterialfv(face, GL\_SHININESS, value); |

valueには、0.0～128.0の値を指定します。

環境光反射成分を設定するには、以下を行います。

|  |
| --- |
| float values[] = { R, G, B, A };  glMaterialfv(face, GL\_AMBIENT, values); |

引数はいずれも前述の通りです。

## スムースシェーディング

ポリゴンの内部は同一色になるため、ポリゴンで構成されたモデルは、ポリゴンがはっきりわかってしまいます。これを「フラットシェーディング」と言います。これに対し、陰影を平滑化することで滑らかに見せる「スムースシェーディング」が用いられています。

フラットシェーディングの設定は、以下で行います。

|  |
| --- |
| glShadeModel(GL\_FLAT); |

スムースシェーディングの設定は、以下で行います。

|  |
| --- |
| glShadeModel(GL\_SMOOTH); |

## 例:シェーディング

シェーディングを行うプログラムの例を示します。

シェーディングを行うプログラム

|  |
| --- |
| #include <GL/gl.h>  #include <GL/glu.h>  #include <GL/glut.h>  static void display\_callback(void)  {  static float diffuse[] = {0.0, 1.0, 1.0, 1.0};  static float specular[] = {0.8, 0.8, 0.8, 1.0};  static float ambient[] = {0.1, 0.1, 0.1, 1.0};  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, diffuse);  glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, specular);  glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, ambient);  glMaterialf(GL\_FRONT\_AND\_BACK, GL\_SHININESS, 32.0);  glEnable(GL\_DEPTH\_TEST);  glEnable(GL\_LIGHTING);  glutSolidSphere(1.0, 20.0, 20.0);  glDisable(GL\_LIGHTING);  glDisable(GL\_DEPTH\_TEST);  glutSwapBuffers();  glFlush();  }  static void special\_callback(int key, int x, int y)  {  switch (key) {  case GLUT\_KEY\_LEFT:  glRotatef(-10, 0.0, 1.0, 0.0);  break;  case GLUT\_KEY\_RIGHT:  glRotatef(10, 0.0, 1.0, 0.0);  break;  case GLUT\_KEY\_UP:  glRotatef(10, 1.0, 0.0, 0.0);  break;  case GLUT\_KEY\_DOWN:  glRotatef(-10, 1.0, 0.0, 0.0);  break;  case GLUT\_KEY\_PAGE\_UP:  glTranslatef(0.0, 0.0, 0.2);  break;  case GLUT\_KEY\_PAGE\_DOWN:  glTranslatef(0.0, 0.0, -0.2);  break;  }  glutPostRedisplay();  }  static void keyboard\_callback(unsigned char key, int x, int y)  {  switch (key) {  case 's':  glShadeModel(GL\_SMOOTH);  glutPostRedisplay();  break;  case 'f':  glShadeModel(GL\_FLAT);  glutPostRedisplay();  break;  case '\e':  exit(0);  }  }  int main(int argc, char \*argv[])  {  glutInit(&argc, argv);  glutInitWindowPosition(0, 0);  glutInitWindowSize(320, 320);  glutInitDisplayMode(GLUT\_RGBA | GLUT\_DEPTH | GLUT\_DOUBLE);  glutCreateWindow(argv[0]);  glClearColor(0.0, 0.0, 0.0, 1.0);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(45.0, 1.0, 2.0, 10.0);  glTranslatef(0.0, 0.0, -4.0);  glEnable(GL\_LIGHT0);  glutKeyboardFunc(keyboard\_callback);  glutSpecialFunc(special\_callback);  glutDisplayFunc(display\_callback);  glutMainLoop();  return 0;  } |

main()関数では、glEnable(GL\_LIGHT0)によりデフォルトの光源を有効にしています。

ディスプレイのコールバック関数display\_callback()では、表面属性を設定した後、glutSolidSphere()関数で球を描画しています。引数は、半径、緯度および経度方向のポリゴンによる分割数です。また、これがシェーディングの対象となるため、glEnable(GL\_LIGHTING)とglDisable(GL\_LIGHTING)で囲っています。

さらに、一般キーのコールバック関数keyboard\_callback()では、[S]および[F]キーが押された時、それぞれスムースシェーディングおよびフラットシェーディングに設定します。

これをコンパイルして実行すると、図 10のウィンドウが表示されます。

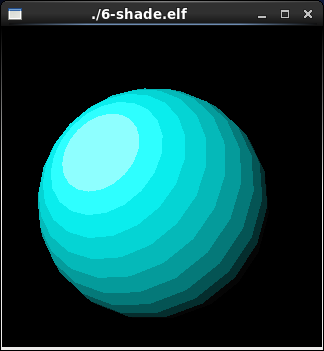
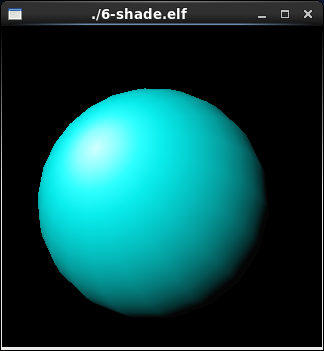


図 10シェーディングプログラムの実行例

### [演習]

表面属性を変更して、色や質感が変わることを確認しましょう。

# 光源設定

## 光源の種類と設定

光源には、主に下記が利用できます(図 11)。いずれも、拡散反射、鏡面反射および環境光反射による属性を設定します。

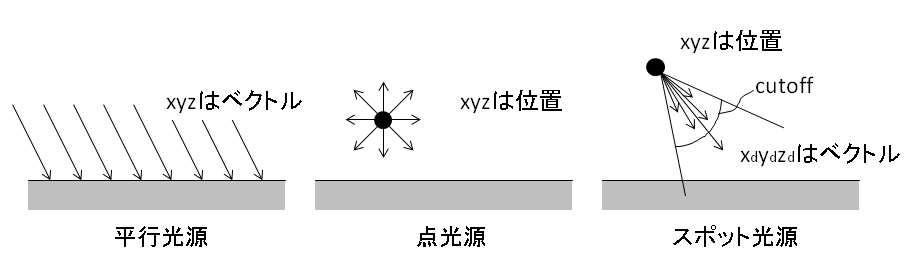


図 11光源の種類

「平行光源」は、太陽光のように無限遠点に光源が存在し、光が平行に到達する光源です。下記のように設定します。

|  |
| --- |
| float light[] = { x, y, z, 0 };  glLightfv(GL\_LIGHTx, GL\_POSITION, light); |

設定する光源のA(Alpha)が0のとき、平行光源となります。x,y,zは光源のベクトルです。

「点光源」は、裸電球のように全方向に空間を照らす光源です。下記のように設定します。

|  |
| --- |
| float light[] = { x, y, z, 1 };  glLightfv(GL\_LIGHTx, GL\_POSITION, light); |

設定する光源のA(Alpha)が1のとき、点光源となります。x,y,zは光源の位置です。

「スポット光源」は、方向性を持った点光源です。下記のように設定します。

|  |
| --- |
| float light[] = { x, y, z, 1 };  float direction[] = { xd, yd, zd };  glLightfv(GL\_LIGHTx, GL\_POSITION, light);  glLightfv(GL\_LIGHTx, GL\_SPOT\_DIRECTION, direction); // 方向  glLightfv(GL\_LIGHTx, GL\_SPOT\_CUTOFF, cutoff); // 広がり  glLightfv(GL\_LIGHTx, GL\_SPOT\_EXPONENT, exponent); // 減衰率 |

x,y,zは光源の位置、xd,yd,zdは光源の方向です。また、cutoffで光源の広がりを、exponentで明るさの減衰率を指定します。

## 例:光源設定

光源設定を行うプログラムの例を示します。

スポット光源の設定を行うプログラム

|  |
| --- |
| #include <GL/gl.h>  #include <GL/glu.h>  #include <GL/glut.h>  static void display\_callback(void)  {  static float diffuse1[] = {0.0, 1.0, 1.0, 1.0};  static float diffuse2[] = {1.0, 0.0, 0.0, 1.0};  static float specular[] = {0.8, 0.8, 0.8, 1.0};  static float ambient[] = {0.2, 0.2, 0.2, 1.0};  float i, j;  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glMaterialfv(GL\_FRONT, GL\_DIFFUSE, diffuse1);  glMaterialfv(GL\_FRONT, GL\_SPECULAR, specular);  glMaterialfv(GL\_FRONT, GL\_AMBIENT, ambient);  glMaterialf(GL\_FRONT, GL\_SHININESS, 128.0);  glEnable(GL\_DEPTH\_TEST);  glEnable(GL\_LIGHTING);  glutSolidSphere(1.0, 20.0, 20.0);  glMaterialfv(GL\_FRONT, GL\_DIFFUSE, diffuse2);  for (i = -3.0; i < 3.0; i += 0.05) {  glBegin(GL\_QUAD\_STRIP);  glNormal3f(0.0, 0.0, 1.0);  for (j = -3.0; j < 3.0; j += 0.05) {  glVertex3f(i, j, -2.0);  glVertex3f(i + 0.05, j, -2.0);  }  glEnd();  }  glDisable(GL\_LIGHTING);  glDisable(GL\_DEPTH\_TEST);  glutSwapBuffers();  glFlush();  }  static void special\_callback(int key, int x, int y)  {  switch (key) {  case GLUT\_KEY\_LEFT:  glRotatef(-10, 0.0, 1.0, 0.0);  break;  case GLUT\_KEY\_RIGHT:  glRotatef(10, 0.0, 1.0, 0.0);  break;  case GLUT\_KEY\_UP:  glRotatef(10, 1.0, 0.0, 0.0);  break;  case GLUT\_KEY\_DOWN:  glRotatef(-10, 1.0, 0.0, 0.0);  break;  case GLUT\_KEY\_PAGE\_UP:  glTranslatef(0.0, 0.0, 0.2);  break;  case GLUT\_KEY\_PAGE\_DOWN:  glTranslatef(0.0, 0.0, -0.2);  break;  }  glutPostRedisplay();  }  static void keyboard\_callback(unsigned char key, int x, int y)  {  switch (key) {  case 's':  glShadeModel(GL\_SMOOTH);  glutPostRedisplay();  break;  case 'f':  glShadeModel(GL\_FLAT);  glutPostRedisplay();  break;  case '\e':  exit(0);  }  }  int main(int argc, char \*argv[])  {  static float diffuse0[] = {0.5, 0.5, 0.5, 1.0};  static float specular0[] = {0.2, 0.2, 0.2, 1.0};  static float position0[] = {1.0, 1.0, 1.0, 0.0};  static float diffuse1[] = {1.0, 1.0, 1.0, 1.0};  static float specular1[] = {1.0, 1.0, 1.0, 1.0};  static float position1[] = {0.0, 4.0, 2.0, 1.0};  static float direction1[] = {0.0, -1.0, -0.5};  glutInit(&argc, argv);  glutInitWindowPosition(0, 0);  glutInitWindowSize(320, 320);  glutInitDisplayMode(GLUT\_RGBA | GLUT\_DEPTH | GLUT\_DOUBLE);  glutCreateWindow(argv[0]);  glClearColor(0.0, 0.0, 0.0, 1.0);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(45.0, 1.0, 2.0, 20.0);  glTranslatef(0.0, 0.0, -6.0);  glLightfv(GL\_LIGHT0, GL\_DIFFUSE, diffuse0);  glLightfv(GL\_LIGHT0, GL\_SPECULAR, specular0);  glLightfv(GL\_LIGHT0, GL\_POSITION, position0);  glEnable(GL\_LIGHT0);  glLightfv(GL\_LIGHT1, GL\_DIFFUSE, diffuse1);  glLightfv(GL\_LIGHT1, GL\_SPECULAR, specular1);  glLightf(GL\_LIGHT1, GL\_SPOT\_EXPONENT, 0.0);  glLightf(GL\_LIGHT1, GL\_SPOT\_CUTOFF, 30.0);  glLightfv(GL\_LIGHT1, GL\_POSITION, position1);  glLightfv(GL\_LIGHT1, GL\_SPOT\_DIRECTION, direction1);  glEnable(GL\_LIGHT1);  glutKeyboardFunc(keyboard\_callback);  glutSpecialFunc(special\_callback);  glutDisplayFunc(display\_callback);  glutMainLoop();  return 0;  } |

main()関数では、GL\_LIGHT0とGL\_LIGHT1の光源を設定しています。GL\_LIGHT0に対しては、光源の属性のみ設定し、デフォルトの平行光源のまま使用しています。GL\_LIGHT1はスポット光源です。

ディスプレイのコールバック関数display\_callback()では、球と平面を描画しています。球の場合、法線ベクトルの設定はglutSolidSphere()に含まれるため行っていませんでした。ですが、平面に対しては明示的に行う必要があります。

これをコンパイルして実行すると、図 12のウィンドウが表示されます

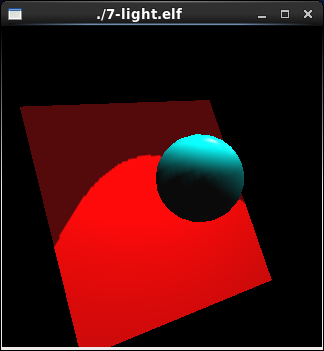


図 12光源設定プログラムの実行例

### [演習]

光源の設定を変えて、どのように表示が変わるか確認してみましょう。

# テクスチャマッピング

## テクスチャマッピングの設定

「テクスチャマッピング」は、画像データを物体の表面に貼りつける方法のことです。木目などの画像を物体に用いることで、リアルなCG画像を表現できます。

まず、ラップ方法を指定します。テクスチャよりも物体表面の方が大きいとき、テクスチャを繰り返すか、最後のピクセル値を用いるかを、下記のように指定します。

|  |
| --- |
| glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, param);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, param); |

paramには、繰り返しを示す「GL\_REPEAT｣か、最後のピクセル値を示す「GL\_CLAMP」を指定します。

次に、テクスチャ環境設定を行います。色の混合を下記のように指定します。

|  |
| --- |
| glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, param); |

paramには、以下のいずれかを指定します。

|  |  |
| --- | --- |
| **設定値** | **概要** |
| GL\_MODULATE | テクスチャと物体の色を掛け算した色に設定 |
| GL\_DECAL | ステッカーを貼るイメージ(透明部分が混合色) |
| GL\_REPLACE | テクスチャの色のみ |
| GL\_BLEND | 別で指定した色と物体の色の混合 |

GL\_BLENDで参照する色の指定は、下記で行います。

|  |
| --- |
| float color[] = { R, G, B, A };  glTexEnvfv(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_COLOR, color); |

テクスチャで使用する画像の定義は、下記で行います。

|  |
| --- |
| unsigned char image[HEIGHT][WIDTH][4]; // RGBAの4つ  glTexImage2D(GL\_TEXTURE\_2D, 0, 4, WIDTH, HEIGHT, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, image); |

定義する画像が2次元のため、GL\_TEXTURE\_2Dを指定します。次の0は解像度が1種類であることを示しています。4はRGBAの4つ、WIDTHおよびHEIGHTは画像の大きさ、GL\_RGBAは画像データのフォーマット、GL\_UNSIGNED\_BYTEはデータの型、最後にデータのポインタを示します。

テクスチャと物体が同じ大きさでない場合、テクスチャのピクセル値をどのように参照するか、指定できます。この拡大・縮小フィルタの設定は、下記のように行います。

|  |
| --- |
| glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, param);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, param); |

GL\_TEXTURE\_MAG\_FILTERで拡大時の、GL\_TEXTURE\_MIN\_FILTERで縮小時の設定を行います。paramには、近いピクセル値を使用する「GL\_NEAREST」か、近傍の平均を使用する「GL\_LINEAR」を指定します。後者は平均化されるため、滑らかにもぼやけたようにも見えます。

最後に、マッピングを行います(図 13)。

|  |
| --- |
| glEnable (GL\_TEXTURE\_2D);  glBegin(GL\_QUADS);  glTexCoord2f(s1, t1); glVertex3f(x1, y1, z1);  glTexCoord2f(s1, t1); glVertex3f(x1, y1, z1);  …  glEnd();  glDisable(GL\_TEXTURE\_2D); |

glTexcoord2f()でテクスチャの頂点、glVertex3f()で物体の頂点を対応付けます。

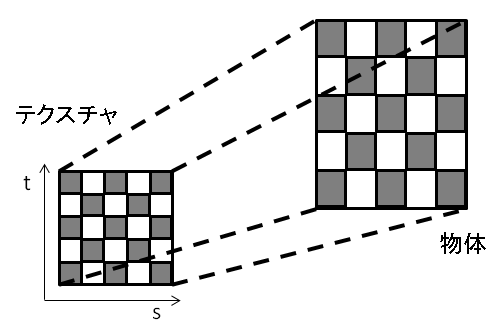


図 13テクスチャマッピング

## 例:テクスチャマッピング

テクスチャマッピングを行うプログラムの例を示します。

テクスチャマッピングを行うプログラム

|  |
| --- |
| #include <GL/gl.h>  #include <GL/glu.h>  #include <GL/glut.h>  #define TEX\_S (8)  #define TEX\_T (8)  #define TEX\_FMT (4)  static unsigned char image[TEX\_T][TEX\_S][TEX\_FMT];  static float repeat = 5.0;  static void init\_image(void)  {  int i, j;  for (i = 0; i < TEX\_T; i++) {  for (j = 0; j < TEX\_S; j++) {  image[i][j][0] = image[i][j][1] = image[i][j][2] =  ((i<4 && j<4) || (i>3 && j>3)) ? 0xff:0;  image[i][j][3] = 0x7f;  }  }  }  static void display\_callback(void)  {  static float diffuse[] = {1.0, 0.0, 0.0, 1.0};  static float specular[] = {1.0, 1.0, 1.0, 1.0};  static float ambient[] = {0.1, 0.1, 0.1, 1.0};  static float texcolor[] = {0.0, 1.0, 1.0, 1.0};  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, diffuse);  glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, specular);  glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, ambient);  glMaterialf(GL\_FRONT\_AND\_BACK, GL\_SHININESS, 128.0);  glEnable(GL\_DEPTH\_TEST);  glEnable(GL\_LIGHTING);  glNormal3f(0.0, 0.0, 1.0);  glTexEnvfv(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_COLOR, texcolor);  glEnable(GL\_TEXTURE\_2D);  glBegin(GL\_QUADS);  glTexCoord2f(0.0, 0.0);  glVertex3f(-2.0, -2.0, 0.0);  glTexCoord2f(0.0, repeat);  glVertex3f(-2.0, 2.0, 0.0);  glTexCoord2f(repeat, repeat);  glVertex3f(2.0, 2.0, 0.0);  glTexCoord2f(repeat, 0.0);  glVertex3f(2.0, -2.0, 0.0);  glEnd();  glDisable(GL\_TEXTURE\_2D);  glDisable(GL\_LIGHTING);  glDisable(GL\_DEPTH\_TEST);  glutSwapBuffers();  glFlush();  }  static void special\_callback(int key, int x, int y)  {  switch (key) {  case GLUT\_KEY\_LEFT:  glRotatef(-10, 0.0, 1.0, 0.0);  break;  case GLUT\_KEY\_RIGHT:  glRotatef(10, 0.0, 1.0, 0.0);  break;  case GLUT\_KEY\_UP:  glRotatef(10, 1.0, 0.0, 0.0);  break;  case GLUT\_KEY\_DOWN:  glRotatef(-10, 1.0, 0.0, 0.0);  break;  case GLUT\_KEY\_PAGE\_UP:  glTranslatef(0.0, 0.0, 0.2);  break;  case GLUT\_KEY\_PAGE\_DOWN:  glTranslatef(0.0, 0.0, -0.2);  break;  }  glutPostRedisplay();  }  static void keyboard\_callback(unsigned char key, int x, int y)  {  switch (key) {  case '+':  repeat += 1.0;  break;  case '-':  repeat -= (repeat > 1.0) ? 1.0:0.0;  break;  case 'n':  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);  break;  case 'l':  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);  break;  case 'm':  glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);  break;  case 'd':  glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_DECAL);  break;  case 'b':  glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_BLEND);  break;  case 'r':  glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_REPLACE);  break;  case '\e':  exit(0);  }  glutPostRedisplay();  }  int main(int argc, char \*argv[])  {  glutInit(&argc, argv);  glutInitWindowPosition(0, 0);  glutInitWindowSize(320, 320);  glutInitDisplayMode(GLUT\_RGBA | GLUT\_DEPTH | GLUT\_DOUBLE);  glutCreateWindow(argv[0]);  glClearColor(0.0, 0.0, 0.0, 1.0);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(45.0, 1.0, 2.0, 20.0);  glTranslatef(0.0, 0.0, -6.0);  glEnable(GL\_LIGHT0);  init\_image();  glTexImage2D(GL\_TEXTURE\_2D, 0, TEX\_FMT, TEX\_S, TEX\_T, 0, GL\_RGBA,  GL\_UNSIGNED\_BYTE, image);  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);  glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_BLEND);  glutKeyboardFunc(keyboard\_callback);  glutSpecialFunc(special\_callback);  glutDisplayFunc(display\_callback);  glutMainLoop();  return 0;  } |

main()関数では、init\_image()を呼び出してテクスチャ画像を作成します。次に、テクスチャの各設定を行っています。

ディスプレイのコールバック関数display\_callback()では、平面を描画しています。テクスチャの頂点と平面の頂点を対応付けています。

また、一般キーのコールバック関数keyboard\_callback()では、[+][-]キーでテクスチャの数、[N][L]キーで拡大・縮小フィルタ、[M][D][B][R]キーでテクスチャ環境設定を行います。

これをコンパイルして実行すると、図 14のウィンドウが表示されます

### [演習]

設定を変えて、それぞれどのように表示が変わるか確認しましょう。

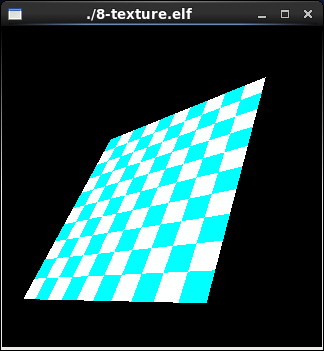


図 14テクスチャマッピングプログラムの実行例

## 自動マッピング

7.1節では、テクスチャとポリゴンの頂点を手動でマッピングしました。これを、自動で行うことができます。下記のように設定します。

|  |
| --- |
| glTexGenf(GL\_S, GL\_TEXTURE\_GEN\_MODE, param);  glTexGenf(GL\_T, GL\_TEXTURE\_GEN\_MODE, param);  glEnable(GL\_TEXTURE\_GEN\_S);  glEnable(GL\_TEXTURE\_GEN\_T);  …描画…  glDisable(GL\_TEXTURE\_GEN\_T);  glDisable(GL\_TEXTURE\_GEN\_S); |

S方向とT方向それぞれに対して、paramでマッピング方法を指定します。方法には以下の種類があります。

|  |  |
| --- | --- |
| **設定値** | **概要** |
| GL\_OBJECT\_LINEAR | オブジェクトに固定された輪郭のマッピング |
| GL\_EYE\_LINEAR | 視点の座標系に固定されたマッピング |
| GL\_SPHERE\_MAP | 平面のテクスチャを物体へ反射させた場合のマッピング |

そして、自動マッピングの対象となる描画をglEnable()～glDisable()で囲みます。

## 例:自動マッピング

テクスチャの自動マッピングを行うプログラムの例を示します。

テクスチャの自動マッピングを行うプログラム

|  |
| --- |
| #include <GL/gl.h>  #include <GL/glu.h>  #include <GL/glut.h>  #define TEX\_S (64)  #define TEX\_T (64)  #define TEX\_FMT (4)  static unsigned char image[TEX\_T][TEX\_S][TEX\_FMT];  static void init\_image(void)  {  int i, j;  for (i = 0; i < TEX\_T; i++) {  int ii = i%8;  for (j = 0; j < TEX\_S; j++) {  int jj = j%8;  image[i][j][0] = image[i][j][1] = image[i][j][2] =  ((ii<4 && jj<4) || (ii>3 && jj>3)) ? 0xff:0;  image[i][j][3] = 0x7f;  }  }  }  static void display\_callback(void)  {  static float diffuse[] = {1.0, 0.0, 0.0, 1.0};  static float specular[] = {1.0, 1.0, 1.0, 1.0};  static float ambient[] = {0.1, 0.1, 0.1, 1.0};  static float texcolor[] = {0.0, 1.0, 1.0, 1.0};  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, diffuse);  glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, specular);  glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, ambient);  glMaterialf(GL\_FRONT\_AND\_BACK, GL\_SHININESS, 128.0);  glEnable(GL\_DEPTH\_TEST);  glEnable(GL\_LIGHTING);  glTexEnvfv(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_COLOR, texcolor);  glEnable(GL\_TEXTURE\_GEN\_S);  glEnable(GL\_TEXTURE\_GEN\_T);  glEnable(GL\_TEXTURE\_2D);  glutSolidTorus(0.4, 0.8, 20.0, 20.0);  glDisable(GL\_TEXTURE\_2D);  glDisable(GL\_TEXTURE\_GEN\_T);  glDisable(GL\_TEXTURE\_GEN\_S);  glDisable(GL\_LIGHTING);  glDisable(GL\_DEPTH\_TEST);  glutSwapBuffers();  glFlush();  }  static void special\_callback(int key, int x, int y)  {  switch (key) {  case GLUT\_KEY\_LEFT:  glRotatef(-10, 0.0, 1.0, 0.0);  break;  case GLUT\_KEY\_RIGHT:  glRotatef(10, 0.0, 1.0, 0.0);  break;  case GLUT\_KEY\_UP:  glRotatef(10, 1.0, 0.0, 0.0);  break;  case GLUT\_KEY\_DOWN:  glRotatef(-10, 1.0, 0.0, 0.0);  break;  case GLUT\_KEY\_PAGE\_UP:  glTranslatef(0.0, 0.0, 0.2);  break;  case GLUT\_KEY\_PAGE\_DOWN:  glTranslatef(0.0, 0.0, -0.2);  break;  }  glutPostRedisplay();  }  static void keyboard\_callback(unsigned char key, int x, int y)  {  switch (key) {  case 'n':  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);  break;  case 'l':  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);  break;  case 'm':  glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);  break;  case 'd':  glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_DECAL);  break;  case 'b':  glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_BLEND);  break;  case 'r':  glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_REPLACE);  break;  case 'o':  glTexGenf(GL\_S, GL\_TEXTURE\_GEN\_MODE, GL\_OBJECT\_LINEAR);  glTexGenf(GL\_T, GL\_TEXTURE\_GEN\_MODE, GL\_OBJECT\_LINEAR);  break;  case 'e':  glTexGenf(GL\_S, GL\_TEXTURE\_GEN\_MODE, GL\_EYE\_LINEAR);  glTexGenf(GL\_T, GL\_TEXTURE\_GEN\_MODE, GL\_EYE\_LINEAR);  break;  case 's':  glTexGenf(GL\_S, GL\_TEXTURE\_GEN\_MODE, GL\_SPHERE\_MAP);  glTexGenf(GL\_T, GL\_TEXTURE\_GEN\_MODE, GL\_SPHERE\_MAP);  break;  case '\e':  exit(0);  }  glutPostRedisplay();  }  int main(int argc, char \*argv[])  {  glutInit(&argc, argv);  glutInitWindowPosition(0, 0);  glutInitWindowSize(320, 320);  glutInitDisplayMode(GLUT\_RGBA | GLUT\_DEPTH | GLUT\_DOUBLE);  glutCreateWindow(argv[0]);  glClearColor(0.0, 0.0, 0.0, 1.0);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(45.0, 1.0, 2.0, 20.0);  glTranslatef(0.0, 0.0, -4.0);  glEnable(GL\_LIGHT0);  init\_image();  glTexImage2D(GL\_TEXTURE\_2D, 0, TEX\_FMT, TEX\_S, TEX\_T, 0, GL\_RGBA,  GL\_UNSIGNED\_BYTE, image);  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);  glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_BLEND);  glTexGenf(GL\_S, GL\_TEXTURE\_GEN\_MODE, GL\_SPHERE\_MAP);  glTexGenf(GL\_T, GL\_TEXTURE\_GEN\_MODE, GL\_SPHERE\_MAP);  glutKeyboardFunc(keyboard\_callback);  glutSpecialFunc(special\_callback);  glutDisplayFunc(display\_callback);  glutMainLoop();  return 0;  } |

main()関数では、マッピング方法をGL\_SPHERE\_MAPに設定しています。

ディスプレイのコールバック関数display\_callback()では、glutSolidTorus()関数によりとーラス(ドーナツ型の物体)を描画しています。引数は、内外の半径と、緯度および経度方向のポリゴンによる分割数です。これを、glEnable()～glDisable()で囲っています。

また、一般キーのコールバック関数keyboard\_callback()では、[O][E][S]キーでマッピング方法を指定しています。

これをコンパイルして実行すると、図 15のウィンドウが表示されます

### [演習]

設定を変えて、それぞれどのように表示が変わるか確認しましょう。

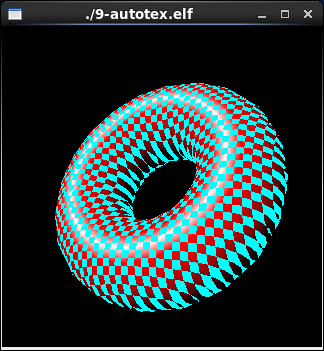


図 15テクスチャの自動マッピングプログラムの実行例

# 参考文献

* 林武文, 加藤清敬:「OpenGLによる3次元CGプログラミング」 コロナ社, 2003
* 床井浩平: 「GLUTによるOpenGL入門―「OpenGL Utility Toolkit」で簡単3Dプログラミング!」 工学社, 2005
* OpenGL Architecture Review Board: 「OpenGL Programming Guide (日本語版)」 アジソン・ウェスレイ, 1993
* GLUTによる「手抜き」OpenGL入門

<http://www.wakayama-u.ac.jp/~tokoi/opengl/libglut.html>

* OpenGL - The Industry
* <http://www.khronos.org/opengl/>
* OpenGL ES - The Standard for Embedded Accelerated 3D Graphics

<http://www.khronos.org/opengles/>

# 変更履歴

|  |  |  |
| --- | --- | --- |
| **版数** | **変更内容** | **年月日** |
| 1.0 | 新規作成 | 2014/MAR/2 |