

平成 20 年度 機械情報工学科演習
コンピュータグラフィクス (2)
OpenGL+GUI(GLUI) による 3DCG の応用
テクスチャマッピング

担当：谷川 智洋 講師，西村 邦裕 助教，中垣 好之 技官
TA：伊東 里香，今井 智章，加倉井 健人

2008 年 10 月 27 日

1 演習の目的

本日の演習では，前回 OpenGL の基礎を元に，OpenGL と GUI と組み合わせを学ぶため，GLUI を用いたプログラミングを行う．さらに，三次元グラフィックスの表示をより高品質にするライティング，テクスチャマッピングについて学習をする．

1.1 資料など

本日の演習の資料などは，

<http://www.cyber.t.u-tokyo.ac.jp/~kuni/enshu2008/>

においてある．

本日使用するソースファイルも同じ場所からダウンロードすること．前回の資料を参照したい場合も同じ場所を参照のこと．

1.2 出席・課題の確認について

出席の確認は，課題の確認によって行う．課題が終了したら，教員・TA を呼び，指示に従って実行して説明せよ．

1.3 「メディアインタフェース」について

次回の「メディアインタフェース」の演習 2008/12/4(木) の際に，EyeToy を持参すること．

2 GLUI

2.1 GLUI とは

GLUI とは、OpenGL で利用可能な簡単なウィジェットライブラリである。以下のような特徴がある。

- OpenGL および glut にのみ依存するため、移植性が高い。OpenGL と glut が動作する C++ の環境であれば、基本的に同じソースコードが動作する。具体的には Linux, Windows (Visual C++, gcc), Macintosh などでも利用可能である。
- glut から自然に移行可能である。glut のソースを大きく変更することなく、ソースに追加していく形で GUI を作成できる。つまりマウスやキーボード、再描画などのイベント（コールバック関数の取り扱いやウィンドウの作成）などほぼ全ての glut の関数がそのまま有効であり、利用可能である。
- GUI のデザインが簡単である。ソースコードの記述順に、ボタンやチェックボックスなどの部品が自動的に（左上から）配置されるため、別途、部品を配置するためのファイルやデータの作成、また専用のデザインツールなどが必要ない。
- 研究活動に向けた機能主体である。立体的な CG を様々な視点から観察する際に便利な、3 次元的な回転や平行移動のための専用の部品を持つ。これを利用すると、直接 `glMultMatrix()` で利用可能な行列が与えられるため、クォータニオンなどを知らなくても対象の回転が簡単に出来る。
- テキスト入力もできる。glut では右クリックによるメニュー程度しかないが、glui を使えばファイル名などのテキストを入力させるなどもスマートに組み込める。glui ではボタンなどが配置できるので、メニューの表示などにより、開発者以外にもわかりやすいプログラムを作ることができる。

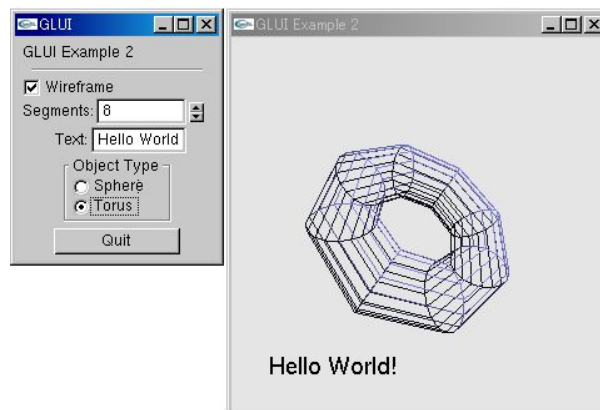


図1 GLUI による表示の例（左側の設定ウィンドウ）

反面、単純さゆえに以下のような欠点がある。ただし、研究で用いる簡単なプログラムには十分である。

- 凝ったデザインが難しい。ソースコードの記述順にボタン等が並ぶだけであるため、自由に部品を配置することが難しい。また、ボタンにアイコンなどの絵を重ねて表示することなどができない。
- 部品の種類が少ない。ウィンドウ端のスクロールバーや、連続値を入力するスライダーなどが用意されていない。

2.1.1 GLUI の概要

GLUI には、以下のような部品が用意されている。

ボタン (Buttons) クリックすることができる押しボタンスイッチ。

チェックボックス (Checkboxes) チェックの ON・OFF。

ラジオボタン (Radio Buttons) グループ内のどれか 1 つにチェックを入れる。

テキストボックス (Editable Text Boxes) ユーザが文字を入力できる白い空白部分。その左側に説明を表示することが可能。

スピナー (Spinners) 上下ボタンで数値が変えられ、数値の部分に直接キーボードから数値を入れることも可能。

リストボックス (Listboxes) 画面中央上にあるように複数の選択肢から 1 つを選ぶ。ラジオボタンよりも選択肢が多い場合や、選択肢の数が動的に変化する場合に使用。

回転コントロール (Rotation Controls) マウスでドラッグすることにより画面に表示された物体を回転させるために用いる。

平行移動コントロール (Translation Controls) 中央下にあるもので、ドラッグすることで対象の平行移動ができる。

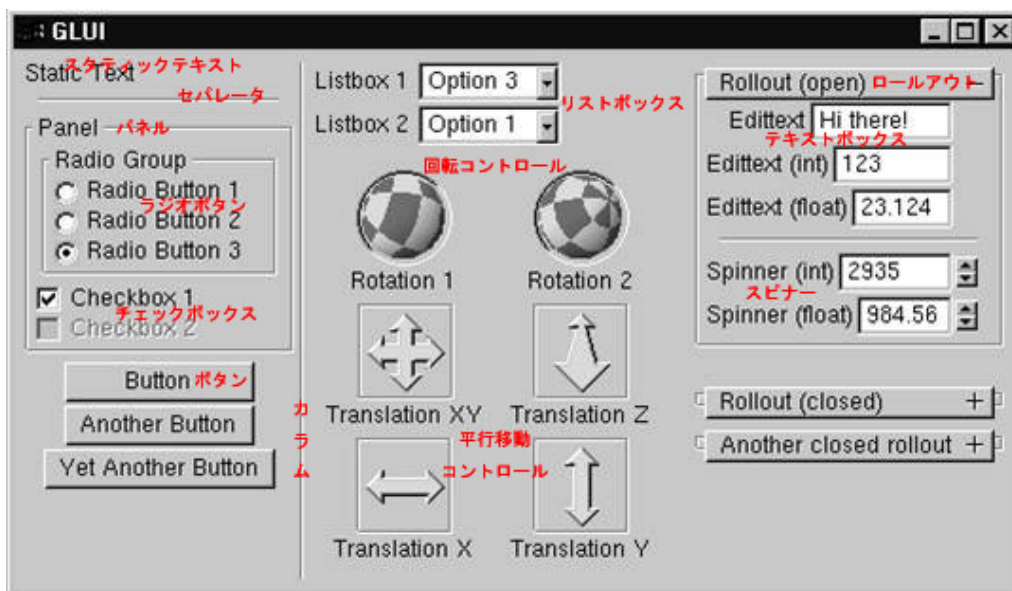


図 2 GLUI の部品一覧

また画面を見やすくしたり、わかりやすくするための以下のような機能もある。

スタティックテキスト (Static Text) 説明文などユーザにより編集しない文字列に使う。プログラム側からは内容を動的に変更することが可能。

セパレーター (Separators) Static Text の下に引いてあるようなもので、区切りに使う。

カラム (Columns) インタフェースが縦に長くなりすぎると見づらい、適宜右へ折り返すような仕組み。線を入れることも (左列と中央列の間)、入れないことも (中央列と右列の間) 可能。

パネル (Panels) ひとまとめの機能をくくる四角形・重ねる (入れ子にする) ことができる。

ロールアウト (Rollouts) ボタン部分を押しと折りたたまれるもので, 図 2 の画面右にあるように最も上のは展開しているが, 下 2 つの部分は畳まれている。

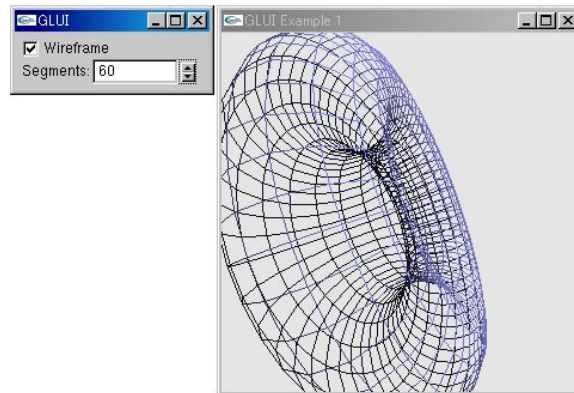


図 3 別ウィンドウとして表示した場合

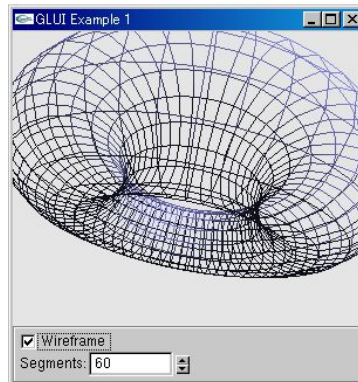


図 4 ウィンドウ内に表示した場合

glui ではこのようなボタンをメインのウィンドウ (自分のプログラムが表示している部分) とは別のウィンドウに出すこともでき (図 3), プログラムの 1, 2 行の変更のみでメインウィンドウの一部と一緒に表示することもできる (図 4)。

別ウィンドウとして表示した場合

```
GLUI *glui = GLUI_Master.create_glui( "GLUI" );
```

ウィンドウ内に表示した場合

```
GLUI *glui = GLUI_Master.create_glui_subwindow(main_window,
                                                GLUI_SUBWINDOW_BOTTOM);
glui->set_main_gfx_window( main_window );
```

GLUTSUBWINDOW_BOTTOM の部分を、GLUTSUBWINDOW_TOP に変えるとウィンドウの上部に、GLUTSUBWINDOW_LEFT に変えるとウィンドウの左側に表示される。

2.2 glut から GLUT

最も簡単な teapot を書くだけのプログラムに UI をつけることを考える。プログラムの必要部分のみ提示する。

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

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);

    glutWireTeapot(1.5);

    glutSwapBuffers();
    glutPostRedisplay();
}

int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
    glutCreateWindow(argv[0]);
    glutDisplayFunc(display);
    glClearColor(0.0, 0.0, 1.0, 0.0);

    glutMainLoop();
    return 0;
}
```

これに GLUT を使ってティーポットの回転ができるようにし、ついでにボタンを押すと終了する様にする。

```
#include <stdio.h>
#include <math.h>
#include <GL/glut.h>
#include <glui.h>
```

```
/*---回転のため---*/
float rotate[16] = {
    1,0,0,0,
    0,1,0,0,
    0,0,1,0,
    0,0,0,1
};

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);

    /*---回転のため---*/
    glPushMatrix();
    glMultMatrixf( rotate );

    glutWireTeapot(1.5);

    glPopMatrix();
    /*---回転のため---*/

    glutSwapBuffers();
    glutPostRedisplay();
}
/*===終了のため===*/
void glutCallback(int num) {
    exit(0);
}
/*===終了のため===*/

int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
    glutCreateWindow(argv[0]);
    glutDisplayFunc(display);
    glClearColor(0.0, 0.0, 1.0, 0.0);
```

```
/*---GLUI initialize---*/
GLUI *glui = GLUI_Master.create_glui("control");

/*---GLUI 回転のため---*/
GLUI_Rotation *view_rot= new GLUI_Rotation(glui, "Rotation",rotate);

/*===GLUI 終了のため===*/
new GLUI_Button(glui, "Exit", 0, gluiCallback);

glutMainLoop();
return 0;
}
```

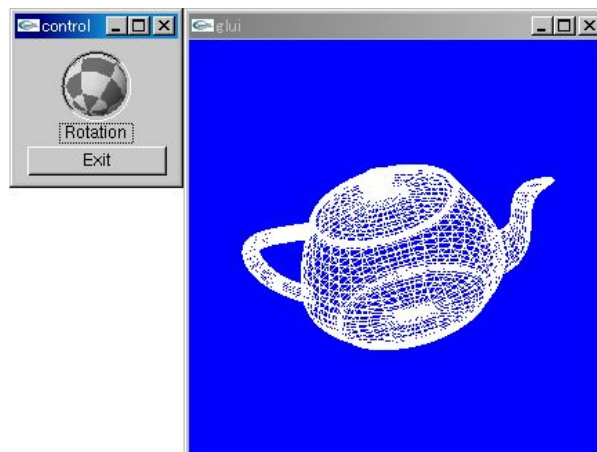


図5 GLUIによるteapotの操作UI

2.3 GLUIのサンプル

GLUIのサンプルはあらかじめフォルダにいてある。example1, example2, example3, example4, example5を参照のこと。実行するには、bin以下のフォルダを実行し、ソースを見るには、example以下のフォルダを参照のこと。

```
/usr/local/src/glui-2.35/src/example/  
/usr/local/src/glui-2.35/src/bin/
```

課題 1

1. `sample_glui.cpp`, `sample_glui_teapot.cpp` および `Makefile` をダウンロードして、実行し、確認せよ。また、Teapot にチェックボックスをつけて `glutWireTeapot` と `glutSolidTeapot` を変更できるようにせよ。
2. `example5`などを参考にして、ボタン、回転コントロール、チェックボックス以外の2つ以上の部品を上記の Teapot に追加せよ。機能は、Teapot の移動やズーム、サイズの指定など何でもよい。
3. 前回作成したロボットアームを UI で制御できるようにせよ。(オプション)

3 画像の取り扱い

OpenGL では、画像において各ピクセルに保存されたカラー (R,G,B,A) を取り扱うことができる。画像のソースには、メモリ内に自動生成した画像データ、レンダリングにより生成された画像、デジカメなどによる画像などを使用出来る。また、画像上に表示するだけでなく、画像はテクスチャ・マップとして使用し、レンダリングされるポリゴンにペーストすることが可能である。

3.1 OpenGL コマンド

ピクセルデータの読み込み

フレーム・バッファからピクセルの方形配列を読み込み、そのデータをメモリに保存する。左下隅が (x, y) に位置し、その左図が width と height で表されるフレームバッファからピクセルデータを読み込み、pixels が示す配列に保存する。

```
void glReadPixels(GLint x, GLint y, GLsizei width, GLsizei height,
                 GLenum format, GLenum type, GLvoid *pixels);
```

format は読み込まれるデータ要素の種類 (RGBA 等) を示し、type は各要素のデータ型式を示す。

フォーマット定数	ピクセルフォーマット
GL_COLOR_INDEX	単一のカラー指標
GL_RGB	赤、緑、青の順序のカラー要素
GL_RED	単一の赤のカラー要素
GL_GREEN	単一の緑のカラー要素
GL_BLUE	単一の青のカラー要素
GL_ALPHA	単一のアルファのカラー要素
GL_LUMINANCE	単一の輝度要素
GL_LUMINANCE_ALPHA	輝度要素とアルファのカラー要素
GL_STENCIL_INDEX	単一のステンシル指標
GL_DEPTH_COMPONENT	単一のデプス要素

表 1 glReadPixels(),glDrawPixels() のピクセルフォーマット

ピクセルデータの書き込み

メモリに保存されたデータから、glRasterPos*() が指定する現在のラスタ位置にあるフレーム・バッファに、ピクセルの方形配列を書き込む。width と height のサイズでピクセルデータの方形を描画する。ピクセル方形の左下隅は現在のラスタ位置になる。

形式定数	データ形式
GL_UNSIGNED_BYTE	符号無し 8 ビット整数
GL_BYTE	符号付き 8 ビット整数
GL_UNSIGNED_SHORT	符号無し 16 ビット整数
GL_SHORT	符号付き 16 ビット整数
GL_UNSIGNED_INT	符号無し 32 ビット整数
GL_INT	符号付き 32 ビット整数
GL_UNSIGNED_BYTE	単精度の浮動小数点

表 2 glReadPixels(),glDrawPixels() のデータ形式

```
void glDrawPixels(GLsizei width, GLsizei height, GLenum format,
                 GLenum type, GLvoid *pixels);

void glRasterPos{234}{sifd}(TYPE x, TYPE y, TYPE z, TYPE w);
void glRasterPos{234}{sifd}v(TYPE *coords);

format と type の意味は, glReadPixels() と同等である .
```

ピクセルデータのコピー

フレーム・バッファの一部から別の部分にピクセルの方形配列をコピーする glReadPixels() の後に glDrawPixels() を呼び出したときと同様であるが、メモリには書き込まれない。

```
void glCopyPixels(GLint x, GLint y, GLsizei width, GLsizei height,
                 GLenum buffer);
```

buffer は GL_COLOR, GL_STENCIL, GL_DEPTH のいずれかのフレームバッファを指定する。

ピクセルデータのパックとアンパック

パックとアンパックは、ピクセルデータがメモリに書き込む、読み込む方法をさす。メモリに保存された画像は、要素と呼ばれるデータのブロックを 1 から 4 つ持っている。このデータは、カラー指標や輝度だけで構成される場合から、各ピクセルに対する赤、緑、青、アルファの要素から構成される場合もある。ピクセルデータの配置方法 format は、各ピクセルに保存される要素の数と順番を蹴っている。

画像データは、通常、方形の 2 次元、または 3 次元配列でメモリに保存されている。多くの場合、配列の部分方形に対応する部分画像の表示、保存が必要である。可能なピクセル格納モードは、glPixelStore*() で制御する。

```
void glPixelStore{if}(GLenum pname, TYPE param);
```

パラメータ `GL_UNPACK*` は、`glDrawPixels()`、`glTexImage1D()`、`glTexImage2D()`、`glTexSubImage1D()`、`glTexSubImage2D()` がデータをメモリからアンパックする方法を制御する。パラメータ `GL_PACK*` は、`glReadPixels()`、`glGetTexImage()` がデータをメモリにパックする方法を制御する。

パラメータ名	形式	初期値	有効範囲
<code>GL_UNPACK_SWAP_BYTES</code> , <code>GL_PACK_SWAP_BYTES</code>	GLboolean	FALSE	TRUEFALSE
<code>GL_UNPACK_LSB_BYTES</code> , <code>GL_PACK_LSB_BYTES</code>	GLboolean	FALSE	TRUEFALSE
<code>GL_UNPACK_ROW_LENGTH</code> , <code>GL_PACK_ROW_LENGTH</code>	GLint	0	任意の非負の整数
<code>GL_UNPACK_SKIP_ROWS</code> , <code>GL_PACK_SKIP_ROWS</code>	GLint	0	任意の非負の整数
<code>GL_UNPACK_SKIP_PIXELS</code> , <code>GL_PACK_SKIP_PIXELS</code>	GLint	0	任意の非負の整数
<code>GL_UNPACK_ALIGNMENT</code> , <code>GL_PACK_ALIGNMENT</code>	GLint	4	1,2,3,4

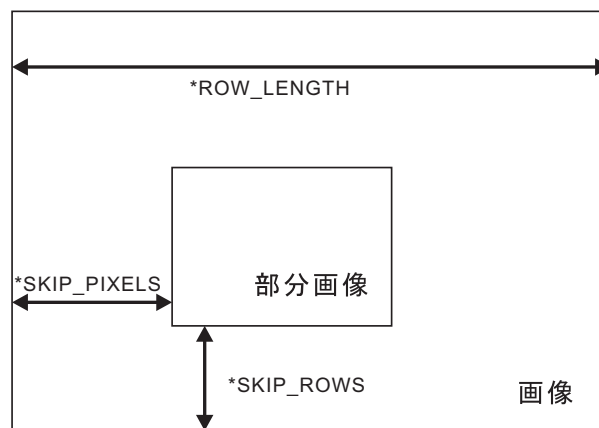


図6 ピクセル格納モード

ピクセルデータの拡大・縮小・線対称変換

```
void glPixelZoom(GLfloat zoom_x, GLfloat zoom_y);
```

ピクセルの書き込み時 (`glDrawPixels()`、`glCopyPixels()`) における拡大、縮小の係数を `x` と `y` のサイズで設定する。初期設定では、`zoom_x` と `zoom_y` は 1.0。どちらも 2.0 の場合各ピクセルは 4 つのピクセルに描画

される。分数による縮小も可能。係数を負にすると、現在のラスタ位置を中心に線対称変換される。

3.2 簡単な例

次の例は、ウィンドウの左下隅にピクセル方形を描画するプログラムである。

glDrawPixels() の使用 : sample-image.c

```
#include <GL/glut.h>
#include <stdlib.h>
#define checkImageWidth 64
#define checkImageHeight 64
GLubyte checkImage[checkImageHeight][checkImageWidth][3];

static GLdouble zoomFactor = 1.0;
static GLint height;

void makeCheckImage(void)
{
    int i, j, c;

    for (i = 0; i < checkImageHeight; i++) {
        for (j = 0; j < checkImageWidth; j++) {
            c = (((i&0x8)==0)^(j&0x8==0))*255;
            checkImage[i][j][0] = (GLubyte) c;
            checkImage[i][j][1] = (GLubyte) c;
            checkImage[i][j][2] = (GLubyte) c;
        }
    }
}

void init(void)
{
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);
    makeCheckImage();
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
}
```

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glRasterPos2i(0, 0);
    glDrawPixels(checkImageWidth, checkImageHeight, GL_RGB,
                 GL_UNSIGNED_BYTE, checkImage);
    glFlush();
}
```

3.3 画像の読み込み

画像をデータとして読み込む関数は標準では用意されていない。そのため、例えば Independent JPEG Group's software の JPEG ライブラリなどを使用する。Linux システムにはインストールされている場合が多い。(http://www.ijg.org/)

JPEG を表示するサンプルプログラム

JPEG ファイルから OpenGL 用のピクセルデータに格納

viewjpeg.c: OpenGL による画像表示プログラム

readjpeg.h, readjpeg.c: JPEG 画像読み込み用プログラム。

testing.jpg: テスト画像

JPEG ライブラリを使用する Makefile の例。

```
TARGET      = viewjpeg
OBJS        = viewjpeg.o readjpeg.o

CC           = gcc
CFLAGS      = -g -Wall -O2
LIBS        = -lglut -lGLU -lGL -ljpeg

TARGET:$(OBJS)
    $(CC) -o $(TARGET) $(OBJS) $(LIBS)

.c.o:
    $(CC) $(FLAGS) -c $<

clean:
    rm -f *.o *~ $(TARGET)
```

4 テクスチャ・マッピング

物体のディテールを表現する一つの方法はポリゴンにより詳細なモデルを構築することであるが、これには膨大な手間を必要とし、ポリゴン数が非常に多くなるため表示のための計算負荷が増大する。これに代わる方法として、物体表面に絵を張りつけることでディテールを表現するテクスチャマッピングの手法が一般的に利用されている。OpenGL ではテクスチャマッピングの機能がサポートされ、これを比較的容易に行うことができる。なお、曲面形状にテクスチャをマップする場合には、マッピングによって生じるテクスチャの歪みやその効果に気をつける必要がある。

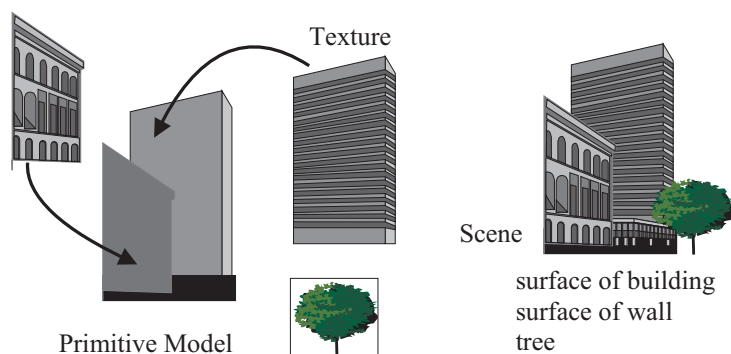


図7 テクスチャ・マッピング技術

4.1 テクスチャマッピングとは

テクスチャマッピングは3次元の平面に画像を貼り込む技術で、建築物や植物など多くのモデルを自然な表現に仕上げるにはこの手法がよく利用される。テクスチャマッピングをするには、画像を配列に読み込み、その画像の位置と面の頂点を対応づける。画像と平面は同一形状である必要はない。

テクスチャマッピングは画像を物体の平面に対応づける。マッピングするときは頂点に画像の位置に対応づける。画像の位置は画像の画素数ではなく、画像全体を0.0...1.0の正規化された座標系で位置を指定する。対応付けには、頂点座標を指定する前に、対応する画像の位置を指定する。

```
void glTexCoord{1,2,3,4}{s,i,d,f}(TYPE coords);
void glTexCoord{1,2,3,4}{s,i,d,f}v(TYPE *coords);
```

4.2 テクスチャ・オブジェクト

OpenGL では、テクスチャの取り扱いにおいて、テクスチャ・オブジェクトという機能を利用する。テクスチャ・オブジェクトはテクスチャ・データを保存し、それが容易に使用できるようにする。多数のテクスチャを制御し、以前にテクスチャ・リソースにロードされているテクスチャに戻ることが可能である。いちいち、画像からロードしなすより、既存のテクスチャオブジェクトをバインド(再利用)する方がほとんどの場合で高速である。テクスチャ・オブジェクトを使用することでテクスチャの適用がもっとも高速になり、パ

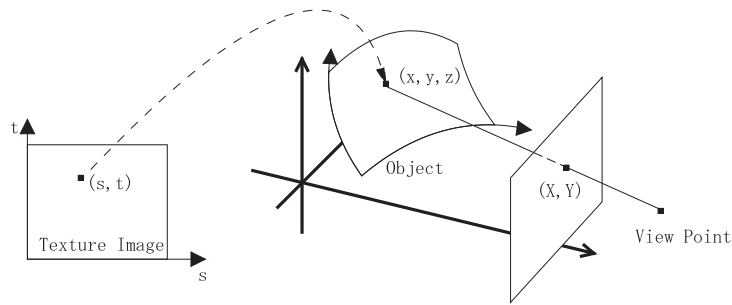


図 8 頂点に画像の座標を対応づける

パフォーマンスも向上する .

1. テクスチャオブジェクトを作成し , それにテクスチャを指定する
2. テクスチャを各ピクセルに適用する方法を指示する
3. テクスチャマッピングを有効化する
4. テクスチャ座標と幾何学座標の双方を与えて , シーンを描画する

OpenGL では , テクスチャにそれぞれ番号をつけて , テクスチャを管理する . 多数のテクスチャを制御し , 以前にテクスチャとしてロードされているテクスチャに戻る事が可能になる .

4.3 簡単な例

次の例 (sample-checker.c) は , テクスチャマッピングのサンプルプログラムである . `glTexImage2D()` の使用している .

```
#include <GL/glut.h>
#include <stdlib.h>
/* Create checkerboard texture */
#define checkImageWidth 64
#define checkImageHeight 64
GLubyte checkImage[checkImageWidth][checkImageHeight][3];

void makeCheckImage(void)
{
    int i, j, c;

    for (i = 0; i < checkImageWidth; i++) {
        for (j = 0; j < checkImageHeight; j++) {
            c = (((i&0x8)==0)^(j&0x8==0))*255;

```

```
        checkImage[i][j][0] = (GLubyte) c;
        checkImage[i][j][1] = (GLubyte) c;
        checkImage[i][j][2] = (GLubyte) c;
    }
}

void myinit(void)
{
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LESS);

    makeCheckImage();
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, checkImageWidth,
                checkImageHeight, 0, GL_RGB, GL_UNSIGNED_BYTE,
                &checkImage[0][0][0]);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
    glEnable(GL_TEXTURE_2D);
    glShadeModel(GL_FLAT);
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-2.0, 1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(0.0, 1.0, 0.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(0.0, -1.0, 0.0);
}
```

```
    glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(2.41421, 1.0, -1.41421);
    glTexCoord2f(1.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);
    glEnd();
    glutSwapBuffers();
}

void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, 1.0*(GLfloat)w/(GLfloat)h, 1.0, 30.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0, 0.0, -3.6);
}

int
main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutCreateWindow("checker");
    myinit();
    glutReshapeFunc (myReshape);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;          /* ANSI C requires main to return int. */
}
```

テクスチャの準備

テクスチャの準備は `initTexture` 関数の中でおこなわれている。テクスチャバッファの2次元配列には、`GLubyte` 型 (`unsigned char`) で `WIDTH × HEIGHT × 4` 個となっている。1画素に `RGBA` の4バイトを割り当てている。テクスチャのサイズは縦横とも2の累乗になっていなければならない、ここでは `64 × 64` になっている。

`initTexture` 関数の内部では、配列にチェック模様を格納していく。白いチェック模様を描くために、配列番号の下位1ビットで白黒を判定している。つまり2で割り切れる場合は1、そうでなければ0を `c` に代入する。最終的に配列には0か255の値が代入される。

テクスチャの格納

配列をテクスチャバッファに読みこむ。テクスチャの格納もプログラム初期化時に `init` 関数にて実行される。

`glPixelStorei()` は、テクスチャバッファに格納される際の配列の並びを指定する。

`image[]` という名前のついた配列は並びが連続している、という指定をして、テクスチャバッファに格納する。

`glTexParameteri`, `glTexParameterf` はテクスチャバッファを利用する際の設定をおこない、テクスチャの反復やクランプ、拡大縮小時の描画に関する制御が可能である。

`glTexImage2D` は、配列の内容をテクスチャバッファに格納する実質的なコマンドである。このコマンドでは以下の引数を持つ。

```
void glTexImage2D(GLenum target, GLint level, GLint component,
                  GLsizei width, GLsizei height, GLint border,
                  GLenum format, GLenum type, const GLvoid *pixels)
```

頻繁に使うパラメータは `width,height,pixel`。 `width,height` はテクスチャバッファのサイズであり、それぞれ2の累乗であることが必要である。 `pixel` にはテクスチャが格納されている配列へのポインタ (ここでは `image`) を指定する。

テクスチャの貼りつけ

テクスチャの切り取り、貼り付け指定は、物体の描画時におこなわれる。ここでは `display()` 関数で行われる。

重要なのは、テクスチャマッピングを有効にするために、`glEnable(GL_TEXTURE_2D)` をコールすることである。`glDisable(GL_TEXTURE_2D)` を呼ぶとテクスチャマッピングが無効になる。この切り替えを上手に行わないと、表示されるポリゴンすべてにテクスチャマッピングが適用される恐れがある。

4.4 よく使う OpenGL の関数

テクスチャオブジェクトの名称設定

```
void glGenTextures(GLsizei n, GLuint *textureNames);
```

`glGenTextures()` は、その番号を取得する関数。n には、取得したい数を入れ、`textureNames` には、配列を返す。この番号をテクスチャ番号として使用する。

テクスチャオブジェクトの作成と使用

```
void glBindTexture(GLenum target, GLuint textureName);
```

使うテクスチャを選択する。target は、`GL_TEXTURE_1D` もしくは `GL_TEXTURE_2D`。textureName には、使おうとするテクスチャの番号を渡す。

テクスチャの指定

```
void glTexImage2D(GLenum target, GLint level, GLint format,
                 GLsizei width, GLsizei height, GLint border,
                 GLenum format, GLenum type, const GLvoid *pixels);
```

2次元テクスチャを定義する。target には、`GL_TEXTURE_2D` を入れる。ミップマップを作るときなどは、`GL_PROXY_TEXTURE_2D` を入れることもある。

level は通常は 0 を指定。自分でミップマップを作る場合、1 以上の値を入れる。

format は読み込まれるデータ要素の種類 (RGBA 等)。画像と同様であり、`GL_RGB` か `GL_RGBA` を指定する場合が多い。

width と height は、テクスチャマップの幅と高さ。OpenGL では、テクスチャの大きさは、2 の累乗である必要がある。すなわち、1, 2, 4, 8, 16, 32, 64, 128, 256...。規格上は最低でも 64x64 の大きさのテクスチャがサポートされ、実際には、1024x1024 ぐらいはサポートされる。

border は、テクスチャに枠をつけるなら 1 を指定する。普通は 0。

format は、pixels に渡すデータの形式。通常は、RGB 形式で記録されている場合、`GL_RGB` を指定。もし、PNG などのフォーマットで透明度設定もあれば、`GL_RGBA` を指定する。

type には、pixels に渡すデータの型を指定。通常は各色 8 ビットなので、`GL_UNSIGNED_BYTE` を指定する。

課題 2

1. sampel-checker.c、sample-texbind.c をダウンロードし、それぞれ実行せよ。テクスチャの取り込みパラメータや貼り付けパラメータを各自で変更し、指定方法が表示にどう影響するか確認せよ。
2. JPEG 読み込みプログラム viewjpeg.c、readjpeg.c などダウンロードし、実行せよ。さらに、EyeToy を持ってきている人は、EyeToy から画像をキャプチャし、表示せよ。EyeToy からのキャプチャのやり方の一つとして、「アプリケーション」「マルチメディア」「XawTV」を起動し、「j」ボタンでキャプチャが可能である。EyeToy がない場合は、異なる画像を表示せよ。
3. sample-image.c を実行し、動作確認せよ。そして、生成する画像イメージの代わりに JPEG イメージを使用し、同様に拡大縮小を可能にせよ。
4. 一枚のポリゴンではなく、いろいろなオブジェクトについても表示してみよ。glut-SolidTeapot が実はテクスチャマッピングに対応している。
5. 数枚以上のテクスチャを読み込み、切り替えて表示することにより、パラパラ漫画のように見えるようにせよ。連続写真は、EyeToy を使って、うまくキャプチャすると良い。(オプション)

課題 3

この課題は、本日学習した GLUI とテクスチャマッピングの両方が含まれている。まず、solar_system_ex.tar.gz をダウンロードして解凍せよ。

1. plant_view 以下の main.cpp の課題となっている部分を修正し実行可能にせよ。
2. orbit_view 以下の main.cpp の課題と成っている部分を修正し実行可能にせよ。
 - (a) 時間と共に、惑星が公転・自転をする。Day rate を変更することにより、惑星の運行の速度が変更するようにせよ。
 - (b) また、下図に示す様に惑星の大きさの変更や、表示モードを変更可能にせよ。

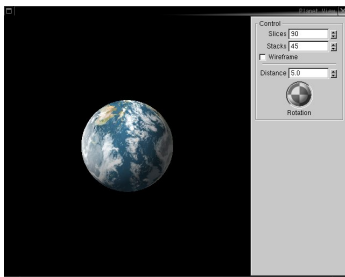


図 9 課題 3.1 実行時の画面

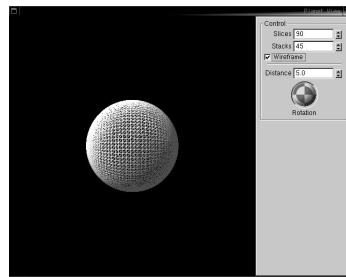


図 10 課題 3.1 実行時の画面

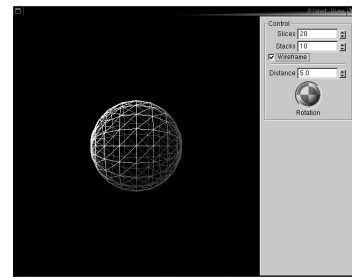


図 11 課題 3.1 実行時の画面

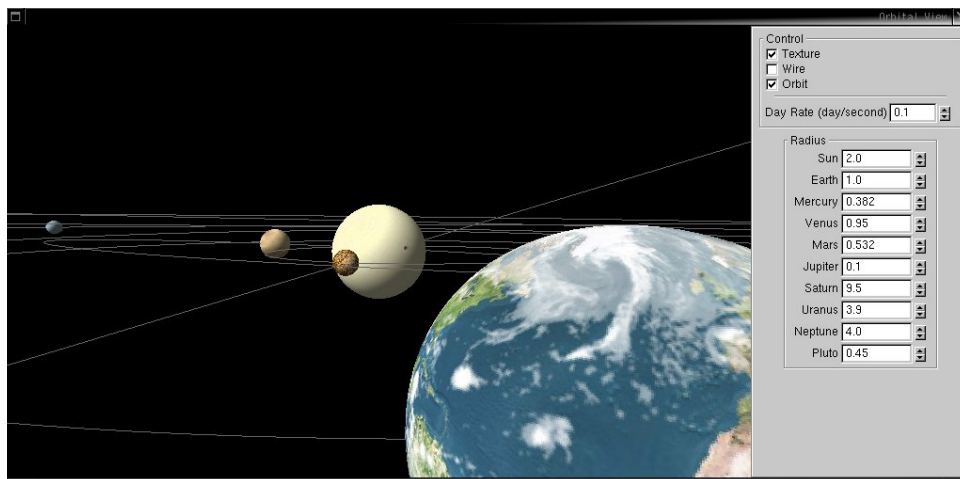


図 12 課題 3.2 実行時の画面

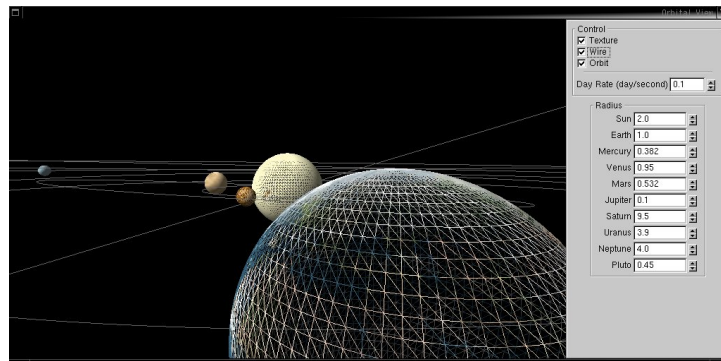


図 13 課題 3.2 実行時の画面 (ワイヤーフレーム)

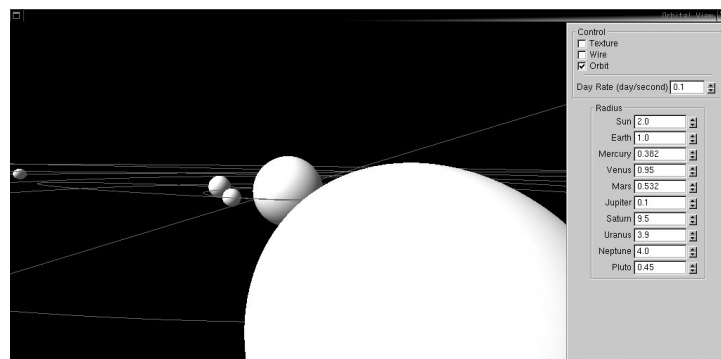


図 14 課題 3.2 実行時の画面 (テクスチャ無し)

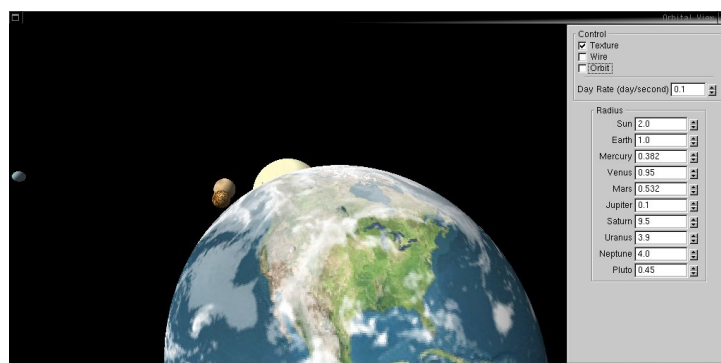


図 15 課題 3.2 実行時の画面 (軌道無し)