CS 450/550 -- Fall Quarter 2025

Project #5

100 Points

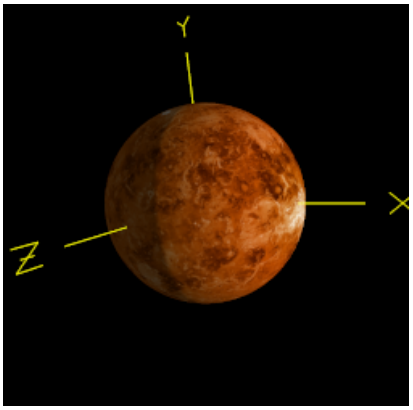
Due: November 13

Texture Mapping

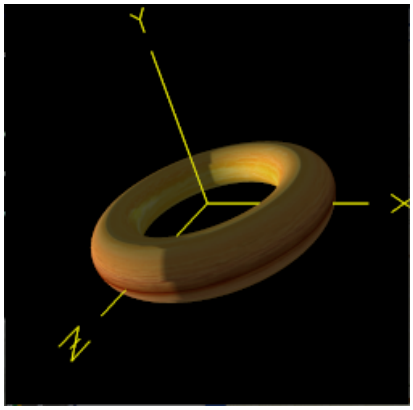
This page was last updated: September 9, 2025

I created a summary document describing the steps in using Texture Mapping. It is called Texturing Steps. Check it out. It might help you on this project!

Introduction



Sphere (venus.bmp)



Torus (saturn.bmp)

OBJ Cow (pluto.bmp)

The goal of this project is to display the Osu* geometric objects and an OBJ object, without textures and with lit textures.

Learning Objective:

When you are done with this assignment, you will understand how to read texture images into your program, how to store them in GPU memory, and how to stretch them onto various objects in your scene. Your future graphics programs will forever be better because you learned this!

Instructions:

Object #	What It Is
0	An OsuSphere
1	An OsuCube
2	An OsuCylinder
3	An OsuCone
4	An OsuTorus
5	An OBJ object

1. Each object needs to have its own unique BMP texture. What you choose is up to you. (Nothing of questionable taste, please.) Save each BMP texture image to your own project area. Be sure each texture image is a BMP file.

2. In your code, read each texture image using the BmpToTexture function and put each texture into a texture object (i.e., bind it to a texture name).

3. Create a display list for each object with the appropriate texture bound and the object drawn.

4. Place a moving point light source somewhere in the scene so that you will be able to demonstrate that your GL_MODULATE texture environment mode works. The exact light source motion is not crucial, so long as it adequately demonstrates dynamic lighting on your textures.

5. Under control of a keyboard hit, change the object being displayed with its texture. Perhaps use the keyboard keys 0-5?

6. Under control of some other keyboard hit, toggle between:
    1. No texture (just a lit object)
    2. Texture image GL_MODULATED'ed

   Perhaps use the 't' key to toggle between these 2 modes?

Planetary Textures, Anyone?

I personally enjoy using NASA's planetary texture images. If you like that too, click here.

A Summary of the Entire Process:

For example:

```
// in the globals:
GLuint  SphereDL, CubeDL, ..., ObjDL;;          // display lists
GLuint  SphereTex, CubeTex, ..., ObjTex;        // texture objects

. . .

// at the end of InitGraphics( ):
        int width, height;
        char *file = (char *)"spheretex.bmp";
        unsigned char *texture = BmpToTexture( file, &width, &height );
        if( texture == NULL )
                fprintf( stderr, "Cannot open texture '%s'\n", file );
        else
                fprintf( stderr, "Opened '%s': width = %d ; height = %d\n", file, width, height );

        glGenTextures( 1, &SphereTex );
        glBindTexture( GL_TEXTURE_2D, SphereTex );
        glPixelStorei( GL_UNPACK_ALIGNMENT, 1 );
        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_LINEAR );
        glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexImage2D( GL_TEXTURE_2D, 0, 3, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, texture );

. . .

// in InitLists( ):
        SphereDL = glGenLists( 1 );
        glNewList( SphereDL, GL_COMPILE );
                SetMaterial( ????? );
                glBindTexture( GL_TEXTURE_2D, SphereTex );      // SphereTex must have already been created when this is called
                OsuSphere( ??, 64, 64 );
        glEndList( );

        . . .

        // THE OBJ OBJECT NEEDS TO BE TREATED DIFFERENTLY FROM THE Osu* OBJECTS:
        GLuint dl = LoadObjMtlFiles( (char *)"dino.obj" );
        ObjDL = glGenLists( 1 );
        glNewList( ObjDL, GL_COMPILE );
                SetMaterial( ????? );
                glBindTexture( GL_TEXTURE_2D, ObjTex );         // ObjTex must have already been created when this is called
                glCallList( dl );                               // a dl can CallList another dl that has been previously created
        glEndList( );

. . .

// in Display( ):
```

```
if( << we-are-in-texture-mode >> )
	glEnable( GL_TEXTURE_2D );
else
	glDisable( GL_TEXTURE_2D );

glEnable( GL_LIGHTING );
glEnable( GL_LIGHT0 );
SetPointLight( GL_LIGHT0, ????? );
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

if( << we-want-to-display-the-sphere >> )
{
	glCallList( SphereDL );
}

. . .

if( << we-want-to-display-the-OBJ-object >> )
{
	glCallList( ObjDL );
}

. . .


glDisable( GL_TEXTURE_2D );
glDisable( GL_LIGHTING );
```

Turn-in:

Use Canvas to turn in:

1. Your .cpp file
2. A short PDF report containing:
    - Your name
    - Your email address
    - Project number and title
    - A description of what you did to get the display you got
    - A cool-looking screen shot from your program
    - The link to the Kaltura video demonstrating that your project does what the requirements ask for. If you can, we'd appreciate it if you'd narrate your video so that you can tell us what it is doing.
3. To see how to turn these files in to Canvas, go to our Project Notes noteset, and go the the slide labeled **How to Turn In a Project on Canvas.**
4. **Be sure that your video's permissions are set to *unlisted.***. The best place to set this is on the OSU Media Server.
5. A good way to test your video's permissions is to ask a friend to try to open the same video link that you are giving us.
6. The video doesn't have to be made with Kaltura. Any similar tool will do.

Files You Might Want to Turn On in Your Sample Code

```
#include "setmaterial.cpp"
#include "setlight.cpp"
#include "osusphere.cpp"
#include "osucube.cpp"
#include "osucylindercone.cpp"
#include "osutorus.cpp"
#include "bmptotexture.cpp"
#include "loadobjmtlfiles.cpp"
//#include "keytime.cpp"
//#include "glslprogram.cpp"
```

Timing Your Scene Animation

Set a constant called something like MS_PER_CYCLE that specifies the number of milliseconds per animation cycle. Then, in your Idle Function, query the number of milliseconds since your program started and turn that into a floating point number between 0. and 1. that indicates how far through the animation cycle you are.

```
// in the globals:
float Time;


// in Animate( ):
	int ms = glutGet( GLUT_ELAPSED_TIME );
	ms %= MS_PER_CYCLE;
	Time = (float)ms / (float)MS_PER_CYCLE;		// [0.,1.)
```

Then use Time to animate the light position.

Grading:

Item	Points
Able to toggle between drawing the 6 objects	20
Draw the objects with the correct textures	30
Lighting the textures works	30
The light source moves	20
Potential Total	100

Sidebar: Much Of This Can Be Automated, But Doesn't Have to Be

Joe Graphics, being somewhat lazy and a lousy typist, always looks for ways to automate repetitive operations into a for-loop. Here is what he did for this project. He first created a struct with all the object information listed:

```
struct object
{
	char*			name;
	char*			file;
	int			displayList;
	char			key;
	unsigned int		texObject;
};
```

He then populated an array of that struct like this:

```
struct object Objects[ ] =
{
	{ "Sphere",	"spheretex.bmp",	0, '0', 0 },
. . .
	{ "Obj",	"objtex.bmp",		0, '5', 0 },
};

const int NUMOBJECTS = sizeof(Objects) / sizeof(struct object);
```

This allowed some setup operations to be put in a for-loop:

```
	// must be done before the obj display list is created:
	GLuint dl = LoadObjMtlFiles((char*)"dino.obj");

	for (int i = 0; i < NUMOBJECTS; i++)
	{
		Objects[i].displayList = glGenLists(1);
		glNewList( Objects[i].displayList, GL_COMPILE );
		glBindTexture( ??? );
		switch( i )
		{
			case 0:
				OsuSphere(1.f, 128, 32);		break;
			case 1:
				OsuCube(1.f);				break;
			case 2:
				OsuCylinder(0.5f, 1.f, 128, 32);	break;
			case 3:
				OsuCone(1.f, .2f, 1., 128, 32);		break;
			case 4:
				OsuTorus(0.25f, 1.f, 128, 128);		break;
			case 5:
```

```
                        glCallList(dl);                         break;
                }
                glEndList();
        }
```

He then created a global variable:

```
// in the globals:
int     NowObject;

. . .

        // in Reset( ):
        NowObject = 0;

. . .


        // in Keyboard( ):
        switch( c )
        {
                case '0':
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                        NowObject = key - '0';  // turn the ascii character into an integer
                        return;
                . . .

. . .

        // in Display( ):
        glCallList( Objects[NowObject].displayList );
```

There is absolutely no requirement that you do it this way! If this makes no sense to you, then just do everything an-object-at-a-time. It will work fine either way.