CS 450/550 -- Fall Quarter 2023

Project #5

100 Points

Due: November 16

Texture Mapping

This page was last updated: September 7, 2023

Introduction

The goal of this project is show the planets at their scaled size, with and without textures, with and without lighting.

Planet	Relative Diameter	.bmp File
Venus	0.95	venus.bmp
Earth	1.00	earth.bmp
Mars	0.53	mars.bmp
Jupiter	11.21	jupiter.bmp
Saturn	9.45	saturn.bmp
Uranus	4.01	uranus.bmp
Neptune	3.88	neptune.bmp

Left-click on a texture file above to see what it looks like.
Right-click on a texture file above to save it in your own file area.
All of these textures have a resolution of 1024x512.

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 quickly stretch them onto various objects in your scene. Really, it was only when texture-mapping hardware appeared in consumer-level graphics hardware that the gaming world took off. Your graphics programs will forever be better because you learned this!

Instructions:

1. Save each texture to your own file area.
2. In your code, read each texture (the BmpToTexture funxtion) and put each texture into a texture object (i.e., bind it to a texture name).
3. Create a display list that holds an OsuSphere with radius=1.0.
4. Create other display lists that hold each planet, properly scaled, and with the appropriate texture bound.
5. Place a moving point light source somewhere in the scene so that you will be able to demonstrate that your GL_MODULATE mode works. The exact light source motion is not crucial, so long as it adequately demonstrates dynamic lighting on your textures.
6. Under control of a keyboard hit, change the planet being displayed, in the correct scale, with the correct texture. Perhaps use the keys 'v', 'e', 'm', 'j', 's', 'u', 'n'.
7. Under control of another keyboard hit, toggle between:
    1. No texture -- just a lit white sphere with the proper scale
    2. Proper scale with the texture image GL_REPLACE'ed
    3. Proper scale with the texture image GL_MODULATED'ed

   Perhaps use the 't' key and have it toggle between these 3 modes.

A Summary of the Entire Process:

For example:

```
// globals:
int	SphereDL, MarsDL;		// display lists
int	MarsTex;			// texture object

. . .

// at the end of InitGraphics( ):
	int width, height;
	char *file = (char *)"mars.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, &MarsTex );
	glBindTexture( GL_TEXTURE_2D, MarsTex );
	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 );
		OsuSphere( 1., ??, ?? );
	glEndList( );

	MarsDL = glGenLists( 1 );
	glNewList( MarsDL, GL_COMPILE );
		glBindTexture( GL_TEXTURE_2D, MarsTex );	// MarsTex must have already been created when this is called
		glPushMatrix( );
			glScalef( 0.53f, 0.53f, 0.53f );	// scale of mars sphere, from the table
			glCallList( SphereDL );		// a dl can call another dl that has been previously created
		glPopMatrix( );
	glEndList( );

. . .

// in Display( ):
	if( << we-are-in-a-texture-mode >> )
		glEnable( GL_TEXTURE_2D );
	else
		glDisable( GL_TEXTURE_2D );

	gleable( GL_LIGHT0 );
	if( << we-are-in-a-lighting-mode >> )
	{
		glEnable( GL_LIGHTING );
		glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
	}
	else
	{
		glDisable( GL_LIGHTING );
		glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
	}

	if( << we-want-to-display-mars >> )
		glCallList( MarsDL );


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

Turn-in:

Use the Teach system to turn in:

1. Your .cpp file
2. A PDF report containing:
    - Project number and title
    - Your name
    - Your email address
    - A description of what you did to get the display you got.
    - A couple of cool-looking screen shots from your program, showing lighting vs. no-lighting and GL_REPLACE vs. GL_MODULATE.
    - The link to the your 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. **Be sure that your video's permission is set to *unlisted*.**. The best place to set this is on the OSU Media Server.
4. 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.
5. 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 "bmptotexture.cpp"
#include "osusphere.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 change the light position.

Something to Notice

Anytime you draw a globe, you will find that orthographic view looks nicer than perspective. This is because you can see more of the sphere that way. Try it.

Grading:

Item	Points
Draw the spheres with the correct scaling	20
Draw the spheres with the correct GL_REPLACE textures	30
Draw the spheres with the correct GL_MODULATE textures	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 planet information listed:

```
struct planet
{
	char*			name;
	char*			file;
	float			scale;
	int			displayList;
	char			key;
	unsigned int		texObject;
};
```

He then populated an array of that struct like this:

```
struct planet Planets[] =
{
	{ "Venus",	"venus.bmp",	0.95f, 0, 'v', 0 },
	{ "Earth",	"earth.bmp",	1.00f, 0, 'e', 0 },
	{ "Mars",	"mars.bmp",	0.53f, 0, 'm', 0 },
	{ "Jupiter",	"jupiter.bmp",	11.21f, 0, 'j', 0 },
	{ "Saturn",	"saturn.bmp",	9.45f, 0, 's', 0 },
	{ "Uranus",	"uranus.bmp",	4.01f, 0, 'u', 0 },
	{ "Neptune",	"neptune.bmp",	3.88f, 0, 'n', 0 },
};

const int NUMPLANETS = sizeof(Planets) / sizeof(struct planet);
```

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

```
for( int i = 0; i < NUMPLANETS; i++ )
{
	<< do something with Planets[i].file >>
}
```

He then created a global variable:

```
intt	NowPlanet;
```

which was initialized to 1 (Earth) in Reset( ) and could be changed from the Keyboard( ) function.
Having that variable allowed him to do things like this:

```
glCallList( Planets[NowPlanet].displayList );
```

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