CSP2307 Introduction to 3D Game Programming

Workshop 5: Texture Mapped Bobcat Rampage!

Experiment with OpenGL texture mapping.

Value: 8 Marks.
Due: In the workshop of the following week.

Outcomes:

- Implement a skybox background and texture mapped objects.

Preparation:

NOTE 1: Getting GL_HEADACHES? If things don't look as they should, check for OpenGL errors:

```
if (glGetError()!=GL_NO_ERROR)
	{
		MessageBox (NULL,"error detected","Error",MB_OK);
	}
```

If the above makes a message box pop up, an error has occurred somewhere before the execution of the glGetError(). This maybe one some line before glGetError, but remember, if in a loop (such as the render loop) it could have been caused by any line after the glGetError() on the previous iteration – to find out exactly where the error is, glGetError is best used in pairs to 'trap' the error.

NOTE 2: If you are using your own image to do the texture mapping – make sure its sized appropriately so that its dimensions correspond to number that are equal to 2 to the power of something (eg. 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024).

NOTE 3: The skybox images for this workshop were created using Terragen, available free for personal or non-commercial use from:
http://www.planetside.co.uk/terragen/
Besides skyboxes, Terragen can also be useful for generating terrain maps and terrain textures for your games.

Preparation

The developers Image Library can be found at http://openil.sourceforge.net/, the code from the last workshop includes the required files so that you can use the library without needing to install anything. Note the location of IL header files in the IL directory and the location of .lib files in the source root directory.

To help in the loading of textures, a class, TextureCreator, has been put on blackboard under this week's material. To keep things simple, the implementation is confined to the header file (TextureCreator.h) rather than being split into a separate .cpp file.

Step 1: Copy the TextureCreator.h file into the root directory of last week's code (don't forget to backup your existing solution).

Step 2: In Visual Studio, right-click on the project (framework) in the Solution Explorer and add a New Filter, give it a name (eg. TextureCreator). Creating a new filter (folder) is optional, but helps to keep large projects organized.

Step 3: Right-click inside the new folder and Add an existing item - choose. TextureCreator.h (note that Filters are virtual folders in Visual Studio, no actual folder has been created on the file system).

Examine the code – the class is simple, containing one static method that takes a file name, loads the corresponding image using the OpenIL library, creates an OpenGL texture object from it, deletes the OpenIL image, and returns the OpenGL handle to the texture object.

The method is declared static, so that you don't need to have an instance of the TextureCreator class to use it. It can be called in your program using:

```
GLuint m_textureObject = TextureCreator::loadTexture("fileName.jpg");
```

NOTE: before you can use the IL library, you need to make a call to ilInit(); This has not been done in the texture loader (you will need to do it somewhere)

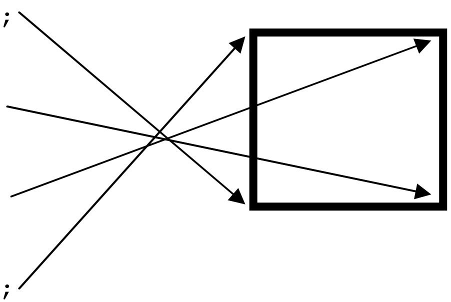
Step 4: Make sure it works – use the TextureCreator to load one of the images in your project (eg. skyboxN.bmp) and map it onto a square.

```
        // draw a square
        glBindTexture (GL_TEXTURE_2D, m_textureObject);
        glColor3f(1,1,1);
        glBegin (GL_QUADS);
              glTexCoord2f (0.0f, 0.0f);
              glVertex3f (-1.0f, 0.0f, 0.0f);

              glTexCoord2f (1.0f, 0.0f);
              glVertex3f (1.0f, 0.0f, 0.0f);

              glTexCoord2f (1.0f, 1.0f);
              glVertex3f (1.0f, 2.0f, 0.0f);

              glTexCoord2f (0.0f, 1.0f);
              glVertex3f (-1.0f, 2.0f, 0.0f);
        glEnd();
```



Task 1 – Skybox (2 Marks)

In outdoor games, how do you draw the surroundings in the distance without drawing a lot of geometry? The simplest answer is a skybox – a cube that moves along with the player, with an image mapped onto it. The images for a skybox were included with the code from last week's workshop.

In this task, you will modify the Geometry class to hold and load a texture for the Geometry object and render it using OpenGL Immediate mode (as opposed to Vertex Buffer Objects).

Step 1: Modify Geometry.h

- Add a member variable to store the texture handle (eg. `GLuint m_material;`)
- Add a string member variable to hold the name of the texture file (eg. `string m_materialFile;`).
- Add one member variable to store a set of texture coordinates, and another to store a set of indices to texture coordinates (you may use the vector type - similar to the way vertex coordinates are stored).
- Add a method to set the name of the texture file – this method should load the corresponding texture into a texture object, holding the handle in m_material.
- Modify the set geometry method so that it takes all the required parameters for a textured object:

```
      void setGeometry( vector<Vertex3>& vertices, vector<int>&
triangles, vector<Vertex3> &textureCoordinates, vector<int>
&textureIndices)
      {
            m_vertexCoordinates = vertices;
            m_triangleIndices = triangles;
            m_textureCoordinates = textureCoordinates;
            m_textureIndices = textureIndices;
            computeBoundingSphere(Vector3f(0,0,0));
      }
```

- Modify the drawOpenGLImmediate() method – if the Geometry has a material file enable texture mapping and bind its texture:

```
if (!m_materialFile.empty())
{
      if (!glIsTexture(m_material))
      {
            m_material = TextureCreator::loadTexture(m_materialFile);
      }
      glEnable(GL_TEXTURE_2D);
      glBindTexture (GL_TEXTURE_2D, m_material);
}
```

- Modify the drawing loop in the drawOpenGLImmediate() method to set a texture coordinate before drawing the vertex:

```
      if (m_material!=NULL)
      {
      glTexCoord2f(m_textureCoordinates[m_textureIndices[count]].x(),
m_textureCoordinates[m_textureIndices[count]].y());
      }
```

- Disable texture mapping after drawing the whole thing:
  `glDisable(GL_TEXTURE_2D);`

- Compile and make sure things are still running! – if not, fix it!

Step 2: Modify the application

In winMain.cpp you will see that the Geometry of the things being drawn is located in a vector called '**worldThings**', which is set up with the objects (cars, houses, etc.) in the method **generateMap**. Since this vector of geometry is checked for collisions etc., putting the skybox geometry in it will be wasteful (the skybox is an 'infinite' distance away, so nothing will collide with it)

- Create a new vector of Geometry objects called **sky** (or similar)
- Similar to the generateMap method, create a generateSkybox method that creates the skybox geometry.
- Create each face of the skybox manually (no model loading) as a separate Geometry object, and use the setGeometry and setMaterial methods of the Geometry class to load the vertex coordinate, texture coordinate, the two index vectors and the name of the material file into it before pushing it into the sky vector.
- In the game loop, pass (by reference) the sky Geometry vector into the renderers Render method (need to modify the method to accept this).
- In the renderers Render method, draw the skybox so that it translates with the player, but does not rotate with them. This is done in the viewing transform:

```
      // **********************************
      // perform the viewing transformation
      // **********************************

      glRotatef(-angleAroundY,0,1,0);

       // enable texture mapping
       glEnable(GL_TEXTURE_2D);
       glPushMatrix();
       glScalef(100,100,100);
      for(vector<Geometry>::iterator skyGeometry = sky.begin();
      skyGeometry<sky.end(); skyGeometry++)
       {
             skyGeometry->drawOpenGLImmediate();
       }
       glPopMatrix();
       glClear(GL_DEPTH_BUFFER_BIT); // clear the depth buffer (draw
over the skybox)

       // enable texture mapping
       glDisable(GL_TEXTURE_2D);

       glTranslatef(-ExcavatorX, -20, -ExcavatorZ);
```

Task 2 – Texture loader (3 Marks).

The loading of model files is done by ASELoader.h (definition of the class) and ASELoader.cpp (implementation of the class) – currently the loading of texture data is not supported – Fix it.

Step 1: Examine the ASE format and ASE loader

As part of this week's workshop files you get a textured version of the house model (and its texture file). Copy those into your projects resources directory.

Open simpleHouse.ASE with a text editor.
The name and path of the texture file is located in the following structure:

```
*MATERIAL_LIST {
        ...
        *MATERIAL 0 {
                ...
                *MAP_DIFFUSE {
                        ...
                        *BITMAP "resources\houseTexture.jpg"
                        ...
                }
        }
}
```

Geometry information is in the following structure:

```
*GEOMOBJECT {
        ...
        *MESH {
                ...
                *MESH_VERTEX_LIST {
                        *MESH_VERTEX   0        -3.4374   0.0660    6.6461
                        *MESH_VERTEX   1         9.2301   0.0660    6.6461
                        *MESH_VERTEX   2        -3.4374   0.0660   -5.4938
                        *MESH_VERTEX   3         9.2301   0.0660   -5.4938
        ...
                        *MESH_VERTEX  23         7.9591  16.8420   -3.0720
                }
                *MESH_FACE_LIST {
                        *MESH_FACE    0:   A:   0 B:   2 C:   3 AB:   ...
                        *MESH_FACE    1:   A:   3 B:   1 C:   0 AB:   ...
                        ...
                        *MESH_FACE   35:   A:  20 B:  22 C:  18 AB:  ...
                }
                ...
                *MESH_TVERTLIST {
                        *MESH_TVERT 0   0.2823    0.6435    0.0000
                        *MESH_TVERT 1   0.2823    0.9045    0.0000
                        *MESH_TVERT 2   0.0100    0.9045    0.0000
                                                  ...
                        *MESH_TVERT 65  0.3410    0.3020    0.0000
                }
                ...
                *MESH_TFACELIST {
                        *MESH_TFACE 0   0         1         2
                        *MESH_TFACE 1   2         3         0
                        ...
                        *MESH_TFACE 35  62        63        60
                }

        }
}
```

Here is what it all means:
***MESH_VERTEX_LIST – a list of vertex coordinates**
***MESH_TVERTLIST – a list of texture coordinates**
***MESH_FACE_LIST – indices into the vertex coordinate list**
***MESH_TFACE_LIST – indices into the texture coordinate list**

For example, the first triangle in the model has the following vertices:

Vertex 1: (x,y,z) = (-3.4374, 0.0660, 6.6461), (s,t) = (0.2823, 0.6435)

Vertex 2: (x,y,z) = (-3.4374, 0.0660, -5.4938), (s,t) = (0.2823, 0.9045)
Vertex 3: (x,y,z) = (9.2301, 0.0660, -5.4938), (s,t) = (0.0100, 0.9045)

IMPORTANT NOTE: Texture coordinates in the ASE format are given with respect to an origin in the top left-hand corner of the texture image. OpenGL texture coordinates use the bottom left corner as the origin. To convert:

s(OpenGL) = s(ASE)
t(OpenGL) = 1-s(ASE)

Step 2: Modify the ASE loader to load textured geometry.

The loader parses the file and loads the *MESH_VERTEX_LIST and *MESH_FACE_LIST. Extend it so that it also loads the *MESH_TVERTLIST and *MESH_TFACE_LIST and the name of the texture file (just the name, the TextureCreator class will load the actual texture).

Step 3: Draw the texture mapped house model using OpenGL Immediate mode.

Task 3 – Texture mapping in VBOs (3 Marks).

Add texture coordinates to your indexed Vertex Buffer Objects and render them with a texture.

***This is much harder than it initially sounds. You have a set of vertex coordinates and a set of texture coordinates, with two vectors of indices into them. For indexed vertex buffer objects, you need to collapse this down to one vector of indices! Here is some (pseudo) code:**

```
vector<Vertex3> expandedVertices;
vector<Vertex3> expandedTextures;
vector<int> expandedIndices; // an index into both Vertex and Texture
int numberOfAddedVertices = 0;

// loop through the existing indices
for (int i = 0; i<m_triangleIndices.size(); i++)
{
      // check if the vertex/texture pair corresponding to:
      // m_triangleIndices[i] m_textureIndices[i] is already
      // pointed to by expandedIndices.

      // if it is, just point to it
      expandedIndices.push_back(…);

      // otherwise insertert the vertex and texture coodinates,
      // then point to them
      expandedVertices.push_back(…);
      expandedTextures.push_back(…);
      expandedIndices.push_back(numberOfAddedVertices);
      numberOfAddedVertices++;
}
```