Using the Nate Robins OpenGL Tutors with Goblin XNA

This is a brief description of how to use the Nate Robins OpenGL Tutors (http://www.xmission.com/~nate/tutors.html) to help gain a better understanding of Goblin XNA. It explains how to map the OpenGL functions used in the tutors to the corresponding features of Goblin XNA. First, download and install the tutors from http://www.xmission.com/~nate/tutors/tutors-win32.zip. Each of the following sections covers one of the tutors.

Transformation
Transformations in a Goblin XNA TransformNode are applied to children in the order scale, rotate, translate (i.e., scale is applied first, then rotation, then translation), independent of the order in which you set those properties. The Transformation tutor demonstrates the same order used in Goblin XNA by default.

<table>
<thead>
<tr>
<th>OpenGL Tutor</th>
<th>Goblin XNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>glTranslatef (x, y, z);</td>
<td>transformNode.Translation = new Vector3 (x, y, z);</td>
</tr>
<tr>
<td>glRotatef (degrees, x, y, z);</td>
<td>transformNode.Rotation = Quaternion.CreateFromAxisAngle (new Vector3 (x, y, z), MathHelper.ToRadians(degrees));</td>
</tr>
<tr>
<td>glScalef (x, y, z);</td>
<td>transformNode.Scale = new Vector3 (x, y, z);</td>
</tr>
</tbody>
</table>

If you want to try the alternative order accessible through the Transformation tutor right-mouse menu, you'll need two TransformNodes to implement it in Goblin XNA. Which properties should you set for each?

Projection
In the Projection Tutor, the right-mouse menu allows you to try projections created with glOrtho and glFrustum that correspond to different frustum geometries than that created with gluPerspective (or the corresponding Goblin XNA calls).

<table>
<thead>
<tr>
<th>OpenGL Tutor</th>
<th>Goblin XNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>gluPerspective (degrees, aspect, zNear, zFar);</td>
<td>camera.FieldOfViewY = MathHelper.ToRadians(degrees);</td>
</tr>
<tr>
<td></td>
<td>camera.AspectRatio = aspect;</td>
</tr>
<tr>
<td></td>
<td>camera.ZNearPlane = zNear;</td>
</tr>
<tr>
<td></td>
<td>camera.ZFarPlane = zFar;</td>
</tr>
<tr>
<td>gluLookAt (eye, center, up);</td>
<td>camera.View = Matrix.CreateLookAt (eye, center, up);</td>
</tr>
<tr>
<td>glOrtho (left, right, bottom, top, near, far);</td>
<td>camera.Projection = Matrix.CreateOrthographicOffCenter (left, right, bottom, top, near, far);</td>
</tr>
<tr>
<td>glFrustum (left, right, bottom, top, near, far);</td>
<td>camera.Projection = Matrix.CreatePerspectiveOffCenter (left, right, bottom, top, near, far);</td>
</tr>
</tbody>
</table>
Light Positioning

Set pos[3] (the last element of pos) to 0.00, which creates a directional light in OpenGL for this tutor. (Recall that \( w=0 \) indicates a direction, while \( w\neq0 \) indicates a position.) Note that OpenGL specifies the direction to the light, while the default shader used in Goblin XNA (like XNA Game Studio and Direct 3D) specifies the direction from the light (in which the light is pointing). Therefore, you will need to negate each of the first three elements of pos to determine the corresponding light direction that would be used in the Goblin XNA default shader.

OpenGL Tutor

```
GLfloat pos[4] = {x, y, z, 0.00};
gluLookAt (eye, center, up);
```

Goblin XNA

```
lightSource.Direction = new Vector3 (-x, -y, -z);
camera.View = Matrix.CreateLookAt (eye, center, up);
```

Light & Material

The illumination and shading model used in the default Goblin XNA shader is simpler than the model used in OpenGL. Lights are directional only (not point or spot light), and neither a light nor a material has an ambient value. This also means that the attenuation constants are not used. However, there is an equivalent to the OpenGL global ambient light color (settable with the right-mouse “Light model parameters” menu entry in the Light & Material Tutor), which interacts with a material’s diffuse color in Goblin XNA. Note that this global ambient color in Goblin XNA is associated with a lightNode, not a lightSource (similar to how it is associated with the glLightModelf function, rather than the glLightfv function, in OpenGL).

In Goblin XNA, the lightSource.Direction is not normalized, unlike OpenGL, in which the direction vector is normalized to unit length. (That is, unlike OpenGL, in the Goblin XNA default shader, the greater the magnitude of the vector, the brighter the light!)

OpenGL Tutor

```
GLfloat light_pos[] = {x, y, z, 0.00};
GLfloat light_Ka[] = {r, g, b, a};
GLfloat light_Kd[] = {r, g, b, a};
GLfloat light_Ks[] = {r, g, b, a};
GLfloat material_Ka[] = {r, g, b, a};
GLfloat material_Kd[] = {r, g, b, a};
GLfloat material_Ks[] = {r, g, b, a};
GLfloat material_Ke[] = {r, g, b, a};
GLfloat lmodel_Ka[] = {r, g, b, a};
```

Goblin XNA

```
lightSource.Direction = new Vector3 (-x, -y, -z);
lightSource.Diffuse = new Vector4 (r, g, b, a);
lightSource.Specular = new Vector4 (r, g, b, a);\nmaterial.Diffuse = new Vector4 (r, g, b, a);
material.Specular = new Vector4 (r, g, b, a);
material.Emissive = new Vector4 (r, g, b, a);
material.SpecularPower = power;
lightNode.AmbientLightColor = new Vector4 (r, g, b, a);
```