**Some Topics Needed for Program 2:**

1. Usings: OpenTK, OpenTK.Graphics.OpenGL
2. Matrix4 – C# class with lots of good methods and overloaded operators (assumes pts on left but stored row-wise, OpenGl assumes column-wise. (AB)T = BTAT

In Visual Studio, right-click on Matrix4 and select “Go to Definition” to see the methods and ops

* Uses operator overloading

1. Orthographic Projection (width, height, front, rear - could have left, right, top, bottom, front, rear

Matrix4 projMat = Matrix4.CreateOrthographic(20.0f, 20.0f, 0.5f, 100.0f);

GL.MatrixMode(MatrixMode.Projection);

GL.LoadMatrix(ref projMat);

1. ModelView Matrix – put Look-At:

Matrix4 lookat = Matrix4.LookAt(eyeX, eyeY, eyeZ, atX, atY, atZ, upX, upY, upZ);

* Eye – where the eye is located
* Look-at – where they are looking at
* Up – which way is up (changes with tilting the head)

GL.MatrixMode(MatrixMode.Modelview);

GL.LoadMatrix(ref lookat);

1. Vertex Buffer Object (VBO) and Vertex Array Object (VAO)

Define all data (that we are going to use) associated with a Vertex

[StructLayout(LayoutKind.Sequential, Pack = 1)]

public struct VertexData

{

public Vector3 Position;

public Vector3 Color;

public Vector3 Normal;

public VertexData(Vector3 pos, Vector3 col, Vector3 norm)

{

Position = pos;

Color = col;

Normal = norm;

}

public VertexData(Vector3 pos, Vector3 col)

{

Position = pos;

Color = col;

Normal = new Vector3(0.0F, 0.0F, 0.0F);

}

}

Assume these variables to create Vertex Buffer Objects (VBOs) and Vertex Array Objects (VAOs).

**We will assume everything is decomposed into triangles.**

private VertexData[] verts; // This is an array of VertexData (see previous page)

private int vboHandle;

private int vaoHandle;

The following 3 steps need to be done once, when the object is created, to set up the VBO and VAO

1. Make the Vertex Buffer Object (VBO) – Allows vertex data to be stored in graphics memory (speed): Generate a buffer, Bind it, and then associate it with the array of verts

GL.GenBuffers(1, out vboHandle); // 1 is the number of buffers to generate

GL.BindBuffer(BufferTarget.ArrayBuffer, vboHandle);

GL.BufferData(BufferTarget.ArrayBuffer,

(IntPtr)(verts.Length \* BlittableValueType.StrideOf(verts)),

verts, BufferUsageHint.StaticDraw);

1. Check to see if the buffer was created properly. Assumes: int size;

GL.GetBufferParameter(BufferTarget.ArrayBuffer, BufferParameterName.BufferSize, out size);

if verts.Length \* BlittableValueType.StrideOf(verts) != size then it failed, give a message

1. Make the Vertex Array Object (VAO): Easier way to specify all the verts in an object.

The steps starting with GL.BindVertexArray(vaoHandle); are the steps that OpenGL will know how to do each time you bind it again and then draw the object.

These steps indicate how the Vertex Object is laid out. **Notice there is no “verts” in here**.

GL.GenVertexArrays(1, out vaoHandle); // 1 is the number of VertArrays to generate

GL.BindVertexArray(vaoHandle); // OpenGL will “remember” the info specified via the steps

// Enable the vertex attributes to be used. We’ll do the normals in a later program.

GL.EnableClientState(ArrayCap.VertexArray);

GL.EnableClientState(ArrayCap.ColorArray);

// Specify the structure, so it knows how to access the data. 3 is the coordinates per point (x,y,z).

GL.VertexPointer(3, VertexPointerType.Float, BlittableValueType.StrideOf(verts), new IntPtr(0));

GL.ColorPointer(3, ColorPointerType.Float, BlittableValueType.StrideOf(verts), new IntPtr(12));

GL.BindVertexArray(0); // Un-bind, so VertArray is no longer associated with the VAO

**This is done each time the object is shown:**

GL.BindVertexArray(vaoHandle); // Bind the VAO. It will “remember” the associations specified above

GL.DrawArrays(PrimitiveType.Triangles, 0, verts.Length); // 0 is starting index in the array, 3rd is count

GL.BindVertexArray(0); // Un-bind, so VertArray is no longer associated with the VAO