Lecture 9: Drawing Models & Primitives with BasicEffect

Prof. Aaron Lanterman
School of Electrical and Computer Engineering
Georgia Institute of Technology

How to peek at the model structure

```csharp
protected override void LoadContent()
{
    gameShip = Content.Load<Model>("ship");
}
```

Bones & Meshes

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Microsoft .NET Framework Graphics Toolkit</th>
<th>Microsoft XNA Framework Graphics Toolkit</th>
</tr>
</thead>
</table>

In meshes: BoundingSpheres & Effects

- BoundingSphere
  - Center: (0.037779, 0.377492, 0.511397) Vector3
  - Radius: 0.02

- BoundingBox
  - Center: (0.027777, 0.377492, 0.511397) Vector3

In meshes: IndexBuffer & VertexBuffer

- IndexBuffer
  - Count: 1

- VertexBuffer
  - Count: 1

- Model

- DrawModel (initialization)

```csharp
private void DrawModel(Model m)
{
    Matrix[] transform = new Matrix[m.Bones.Count];

    for (int i = 0; i < m.Bones.Count; i++)
    {
        Bone bone = m.Bones[i];
        Matrix rotation = bone.TransformBoneTransform.ToMatrix();

        for (int j = 0; j < bone.Transformations.Count; j++)
        {
            Transformation transformation = bone.Transformations[j];
            Matrix matrix = Matrix.CreateRotationX(rotation).
            matrix *= Matrix.CreateRotationY(rotation);
            matrix *= Matrix.CreateRotationZ(rotation);

            // Apply transformations to the bone
            bone.Transformations[j] = matrix;
        }

        transform[i] = bone.TransformBoneTransform.ToMatrix();
    }


    GraphicsDevice.SamplerState = SamplerState.PointClamp;
    GraphicsDevice.DepthStencilState = DepthStencilState.None;

    RenderTargetBuffer = graphics.GraphicsDeviceRenderTargetBuffer;

    foreach (ModelMesh mesh in m.Meshes)
    {
        mesh.Draw(transform, aspectRatio);
    }
}
```

CopyAbsoluteBoneTransformsTo

- Remarks

  In an absolute transform, each bone is transformed according to the position of all parent bones.

  This is the same as iterating the Bones collection and applying the transformation matrix of every parent of a ModelBone to the Transform property of each ModelBone and copying the results into an array that can be indexed into by the bone index.

  An array of transformation matrices for the meshes of any model can be obtained by calling CopyAbsoluteBoneTransformsTo or CopyBoneTransformsTo. The resulting array contains the transforms that describe how each ModelMesh is located relative to one another in the Model. The transformation matrix that should be applied to each ModelMesh can be obtained using the index of the CopyAbsoluteBoneTransformsTo to retrieve a transformation matrix from this collection.
DrawModel (main loop)

```csharp
foreach (ModelMesh mesh in m_Meshes)
{
    foreach (BasicEffect effect in mesh.Effects)
    {
        effect.EnableDefaultLighting();
        effect.View = view;
        effect.Projection = projections;
        effect.World = gameWorldRotation *
                     transform[mesh.ParentBone.Index] *
                     Matrix.CreateTranslation(Position);  
        mesh.Draw();
    }
}
```


How to: use BasicEffect

For our purposes, we're mostly interested in learning how to draw primitives


VertexBuffer

- Contains list of vertices
  - 4-D colors (r,g,b,a)
  - 3-D positions (x,y,z)
  - 2-D texture coordinates (u,v)
  - 3-D normals
- Structures:
  - VertexPositionColor
  - VertexPositionColorTexture
  - VertexPositionNormalTexture
  - VertexPositionTexture
  - Can make your own with VertexElement; see
    "How To: Create and Use a Custom Vertex"

Declare some variables

```csharp
namespace UseBasicEffect
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        Matrix worldMatrix;
        Matrix viewMatrix;
        Matrix projectionMatrix;
        VertexPositionNormalTexture[] cubeVertices;
        VertexDeclaration basicEffectVertexDeclaration;
        VertexBuffer vertexBuffer;
        BasicEffect basicEffect;

        GraphicsDeviceManager graphics;
    }
}
```

Set up cube (1)

```csharp
private void SetupCube()
{
    // Cube vertices
    Vector3 cubeVertices = new Vector3(0, 1, 0); 
    Vector3 cubeVerticesTop = new Vector3(1, 0, 0); 
    Vector3 cubeVerticesBottom = new Vector3(-1, 0, 0); 
    Vector3 cubeVerticesRight = new Vector3(0, 1, 0); 
    Vector3 cubeVerticesLeft = new Vector3(0, -1, 0); 
    Vector3 cubeVerticesFront = new Vector3(0, 0, 1); 
    Vector3 cubeVerticesBack = new Vector3(0, 0, -1); 

    // Cube texture vertices
    Vector3 textureTopLeft = new Vector3(0, 0, 0); 
    Vector3 textureTopRight = new Vector3(1, 0, 0); 
    Vector3 textureTop = new Vector3(0.5, 0.5, 0); 
    Vector3 textureBotLeft = new Vector3(0, 1, 0); 
    Vector3 textureBotRight = new Vector3(1, 1, 0); 
    Vector3 textureBot = new Vector3(0.5, 0.5, 1); 
    Vector3 textureFront = new Vector3(0.5, 0.5, 0); 
    Vector3 textureBack = new Vector3(0.5, 0.5, 1); 

    // Cube normals
    Vector3 cubeNormal = new Vector3(0, 1, 0); 
    Vector3 faceNormal = new Vector3(0.1, 0.1, 0.1); 

    // Cube positions
    Vector3 position = new Vector3(0, 0, 0); 
    Vector3 positionFront = new Vector3(1, 0, 0); 

    // Cube rotation
    Quaternion rotation = Quaternion.Identity; 

    // Cube
    new CubeModel(cubeVertices, cubeVerticesTop, cubeVerticesBottom, cubeVerticesRight, cubeVerticesLeft, cubeVerticesFront, cubeVerticesBack, textureTopLeft, textureTopRight, textureTop, textureBotLeft, textureBotRight, textureBot, textureFront, textureBack, cubeNormal, faceNormal, position, rotation);
}
```


Set up cube (2)

```csharp
// Front face
    cubeVertices[0] = 
        new VertexPositionNormalTexture(
        topLeftFront, frontNormal, textureTopLeft); 
    cubeVertices[1] = 
        new VertexPositionNormalTexture(
        bottomLeftFront, frontNormal, textureBottomLeft); 
    cubeVertices[2] = 
        new VertexPositionNormalTexture(
        bottomRightFront, frontNormal, textureBottomRight); 
    cubeVertices[3] = 
        new VertexPositionNormalTexture(
        topLeftFront, frontNormal, textureTopRight); 
    cubeVertices[4] = 
        new VertexPositionNormalTexture(
        bottomRightFront, frontNormal, textureBottomRight); 
    cubeVertices[5] = 
        new VertexPositionNormalTexture(
        topRightFront, frontNormal, textureTopRight); 
    cubeVertices[6] = 
        new VertexPositionNormalTexture(
        bottomLeftFront, frontNormal, textureBottomLeft); 
```


etc. etc.

Set up basicEffect (instance of BasicEffect)

```csharp
private void InitіationEffect()
{
    // BasicEffect
    BasicEffect = new BasicEffect(graphics.GraphicsDevice, null); 
    BasicEffect.World = Matrix.Identity; 
    BasicEffect.View = Matrix.Identity; 
    BasicEffect.Projection = Matrix.Identity; 
    BasicEffect.VertexFormat = VertexPositionTexture; 
    BasicEffect.Texture = texture; 
    BasicEffect.Diffuse = texture; 
}
```

**PrimitiveType Enumeration**

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LineList</td>
<td>Renders the vertices as a list of isolated straight line segments; the count may be any positive integer.</td>
</tr>
<tr>
<td>LineStrip</td>
<td>Renders the vertices as a single polyline; the count may be any positive integer.</td>
</tr>
<tr>
<td>PointList</td>
<td>Renders the vertices as a collection of isolated points. This value is not supported for indexed primitives.</td>
</tr>
<tr>
<td>TriangleFan</td>
<td>Renders the vertices as a triangle fan.</td>
</tr>
<tr>
<td>TriangleList</td>
<td>Renders the specified vertices as a sequence of isolated triangles; each group of three vertices defines a separate triangle. Back-face culling is affected by the current winding order render state.</td>
</tr>
<tr>
<td>TriangleStrip</td>
<td>Renders the vertices as a triangle strip; the back-face culling flag is flipped automatically on even-numbered triangles.</td>
</tr>
</tbody>
</table>


**Draw lit cube with BasicEffect**

```csharp
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.SteelBlue);
    GraphicsDevice.VertexDeclaration = basicEffect.VertexDeclarations;
    basicEffect.Begin();
    foreach (EffectPass pass in basicEffect.CurrentTechnique.Passes)
    {
        pass.Begin();
        GraphicsDevice.DrawPrimitives(PrimitiveType.TriangleList, 0, 12);
        pass.End();
    }
    basicEffect.End();
    startVertex = primitiveCount;
    basic.Draw(gameTime);
}
```


**DrawIndexedPrimitives example**

- **Example set up:**
  ```csharp
  graphics.GraphicsDevice.Vertices[0].SetSource(vertexBuffer, 0, VertexPositionNormalTexture.SizeInBytes);
  graphics.GraphicsDevice.Indices = lineListIndexBuffer;
  ```

- **Call structure:**
  ```csharp
  public void DrawIndexedPrimitives(PrimitiveType primitiveType, int baseVertex, int minVertexIndex, int numVertices, int startIndex, int primitiveCount)
  ```


**BasicEffect handles 12 different cases**

- Internally set ShaderIndex parameter:
  - 0: Flat color
  - 1: Vertex colors
  - 2: Textured
  - 3: Textured + Vertex colors
  - 4: Lit
  - 5: Lit + Vertex colors
  - 6: Lit + Textured
  - 7: Lit + Textured + Vertex colors
  - 8: Per-pixel lit
  - 9: Per-pixel lit + Vertex colors
  - 10: Per-pixel lit + Textured
  - 11: Per-pixel lit + Textured + Vertex colors

BasicEffect handles 3 directional lights

- Parameters:
  - AmbientLightColor
  - DirLight0Direction
  - DirLight0DiffuseColor
  - DirLight0SpecularColor
  - DirLight1Direction
  - DirLight1DiffuseColor
  - DirLight1SpecularColor
  - DirLight2Direction
  - DirLight2DiffuseColor
  - DirLight2SpecularColor


BasicEffect is just a shader (.fx file)

```csharp
// Texture sampler
uniform const texture BasicTexture;
uniform const sampler TextureSampler : register(s0) = sampler_state {
    Texture = (BasicTexture);
    MipFilter = Linear;
    MinFilter = Linear;
    MagFilter = Linear;
};

// Fog settings
uniform const float FogEnabled : register(c0);
uniform const float FogStart : register(c1);
uniform const float FogEnd : register(c2);
uniform const float3 FogColor : register(c3);
uniform const float3 EyePosition : register(c4); // in world space

// Material settings
uniform const float3 DiffuseColor : register(c5) = 1;
uniform const float Alpha : register(c6) = 1;
uniform const float3 EmissiveColor : register(c7) = 0;
uniform const float3 SpecularColor : register(c8) = 1;
uniform const float SpecularPower : register(c9) = 16;

```

More BasicEffect parameters

- Material properties (colors range 0 to 1)
  - DiffuseColor
  - EmissiveColor
  - SpecularColor
  - SpecularPower
  - Alpha

- Fog properties
  - FogEnabled (0 to disable, 1 to enable)
  - FogStart
  - FogEnd
  - FogColor


BasicEffect is just a shader (.fx file)

```csharp
// Texture sampler
uniform const texture BasicTexture;
uniform const sampler TextureSampler : register(s0) = sampler_state {
    Texture = (BasicTexture);
    MipFilter = Linear;
    MinFilter = Linear;
    MagFilter = Linear;
};

// Fog settings
uniform const float FogEnabled : register(c0);
uniform const float FogStart : register(c1);
uniform const float FogEnd : register(c2);
uniform const float3 FogColor : register(c3);
uniform const float3 EyePosition : register(c4); // in world space

// Material settings
uniform const float3 DiffuseColor : register(c5) = 1;
uniform const float Alpha : register(c6) = 1;
uniform const float3 EmissiveColor : register(c7) = 0;
uniform const float3 SpecularColor : register(c8) = 1;
uniform const float SpecularPower : register(c9) = 16;

```
BasicEffect is just a shader (2)

---

// Vertex shader inputs
---

struct VSInput { 
float4 Position : POSITION;
};

struct VSInputVc { 
float4 Position : POSITION;
float4 Color : COLOR;
};

struct VSInputNm { 
float4 Position : POSITION;
float3 Normal : NORMAL;
};

struct VSInputNmVc { 
float4 Position : POSITION;
float3 Normal : NORMAL;
float4 Color : COLOR;
};

---


BasicEffect is just a shader (3)

---

// Vertex shader outputs
---

struct VertexLightingVSOutput { 
float4 PositionPS : POSITION; // Position in projection space
float4 Diffuse : COLOR0; // Specular.rgb and fog factor
float4 Specular : COLOR1;
};

struct VertexLightingVSOutputTx { 
float4 PositionPS : POSITION; // Position in projection space
float4 Diffuse : COLOR0;
float4 Specular : COLOR1;
float2 TexCoord : TEXCOORD0;
};

struct PixelLightingVSOutput { 
float4 PositionPS : POSITION; // Position in projection space
float4 PositionWS : TEXCOORD0; // diffuse.rgb and alpha
float3 NormalWS : TEXCOORD1;
float4 Diffuse : COLOR0;
};

---


BasicEffect is just a shader (4)

---

// Pixel shader inputs
---

struct VertexLightingPSInput { 
float4 Diffuse : COLOR0;
float4 Specular : COLOR1;
};

struct VertexLightingPSInputTx { 
float4 Diffuse : COLOR0;
float4 Specular : COLOR1;
float2 TexCoord : TEXCOORD0;
};

struct PixelLightingPSInput { 
float4 PositionWS : TEXCOORD0;
float3 NormalWS : TEXCOORD1;
float4 Diffuse : COLOR0;
};

---


BasicEffect is just a shader (5)

---

// Vertex shaders
---

VertexLightingVSOutput VSBasic(VSInput vin) {
    VertexLightingVSOutput vout;
    CommonVSOutput cout = ComputeCommonVSOutput(vin.Position);
    vout.PositionPS = cout.Pos_ps;
    vout.Diffuse = cout.Diffuse;
    vout.Specular = float4(cout.Specular, cout.FogFactor);
    return vout;
}

VertexLightingVSOutput VSBasicVc(VSInputVc vin) {
    VertexLightingVSOutput vout;
    CommonVSOutput cout = ComputeCommonVSOutput(vin.Position);
    vout.PositionPS = cout.Pos_ps;
    vout.Diffuse = cout.Diffuse * vin.Color;
    vout.Specular = float4(cout.Specular, cout.FogFactor);
    return vout;
}

---

BasicEffect is just a shader (6)

```csharp
// Pixel shaders
//-----------------------------------------------------------------------------
float4 PSBasic(VertexLightingPSInput pin) : COLOR {
    float4 color = pin.Diffuse + float4(pin.Specular.rgb, 0);
    color.rgb = lerp(color.rgb, FogColor, pin.Specular.w);
    return color;
}
float4 PSBasicTx(VertexLightingPSInputTx pin) : COLOR {
    float4 color = tex2D(TextureSampler, pin.TexCoord) * pin.Diffuse + float4(pin.Specular.rgb, 0);
    color.rgb = lerp(color.rgb, FogColor, pin.Specular.w);
    return color;
}
float4 PSBasicPixelLighting(PixelLightingPSInput pin) : COLOR {
    float3 posToEye = EyePosition - pin.PositionWS.xyz;
    float3 N = normalize(pin.NormalWS);
    float3 E = normalize(posToEye);
    ColorPair lightResult = ComputePerPixelLights(E, N);
    float4 diffuse = float4(lightResult.Diffuse * pin.Diffuse.rgb, pin.Diffuse.a);
    float4 color = diffuse + float4(lightResult.Specular, 0);
    color.rgb = lerp(color.rgb, FogColor, pin.PositionWS.w);
    return color;
}
```

Syntactic sugar & maybe other bookkeeping

- Using BasicEffect:
  ```csharp
  basiceffect.World = Matrix.Identity;
  ```

- Using a standard user-made Effect:
  ```csharp
effect.Parameters["World"].SetValue(Matrix.Identity);
  ```

- PreferPerPixelLighting, TextureEnabled, VertexColorEnabled, etc. properties set ShaderIndex for you

BasicEffect is just a shader (7)

```csharp
int ShaderIndex = 0;
VertexShader VSArray[12] = {
    compile vs_1_1 VSBasic(),
    compile vs_1_1 VSBasicVc(),
    etc.
    compile vs_1_1 VSBasicPixelLightingNmTx(),
    compile vs_1_1 VSBasicPixelLightingNmTxVc(),
};
PixelShader PSArray[12] = {
    compile ps_1_1 PSBasic(),
    compile ps_1_1 PSBasic(),
    etc.
    compile ps_2_0 PSBasicPixelLightingTx(),
    compile ps_2_0 PSBasicPixelLightingTx(),
};
Technique BasicEffect {
    Pass {
        VertexShader = (VSArray[ShaderIndex]);
        PixelShader = (PSArray[ShaderIndex]);
    }
}
```

Using BasicEffect:
```csharp
effect = new BasicEffect();
effect.World = Matrix.Identity;
```