3D Intro

Slides built from Carter Chapter 4
Vertices

- Objects are made up of 3D points called vertices.
- A Vertex is a point in space.
- In 3D it has 3 coordinates, \((X,Y,Z)\)
- We can generate vertices, design them, or load them
  - Getting an artist to create our content can be easy
  - Building procedurally is more powerful
  - We’ll start by hardcoding some points
Vectors

- In XNA have 3 classes
  - Vector2, Vector3, Vector4 (with 2, 3 and 4 components)
- Mathematically, a vector is a direction, and a point is a location.
  - Vector + vector = vector;
  - Vector + point = point;
  - Point - point = vector;
- In XNA vector and points treated the same
  - What happens when I translate a vector or a point?
Matrices

- In XNA, matrices are 4x4 2d float arrays
- In graphics (and XNA) matrices hold transformations
  - Geometric (from 3d to 3d)
    - Rotation, scaling, translation, shearing
    - From one space to another
      - Object to world
  - Projections (from 3d to 2d plane)
    - Parallel, perspective
    - Camera
- We can perform multiple transforms by multiplying their matrices together.
Creating a camera

- We need some matrices in our game class
  ```csharp
  private Matrix projection;
  private Matrix view;
  private Matrix world;
  ```
- Create InitializeCamera(), call in LoadContent
private void InitializeCamera()
{
    float aspectRatio =
        (float)graphics.GraphicsDevice.Viewport.Height;
    Matrix.CreatePerspectiveFieldOfView(
        MathHelper.PiOver4, aspectRatio, 0.001f, 1000.0f,
        out projection);
    Matrix.CreateLookAt(ref cameraPosition, 
        ref cameraTarget, ref cameraUpVector, out view);
}
CreatePerspectiveFieldOfView

```java
public static Matrix CreatePerspectiveFieldOfView(
    float fieldOfView,
    float aspectRatio,
    float nearPlaneDistance,
    float farPlaneDistance
)
```

use BoundingFrustum
Matrix.CreateLookAt

class Matrix

public static void CreateLookAt(
    ref Vector3 cameraPosition,
    ref Vector3 cameraTarget,
    ref Vector3 cameraUpVector,
    out Matrix result
)
For the view matrix

```csharp
private Vector3 cameraPosition = new Vector3(0.0f,0.0f,3.0f);
private Vector3 cameraTarget = Vector3.Zero;
private Vector3 cameraUpVector = Vector3.Up;
Matrix.CreateLookAt(ref cameraPosition, ref cameraTarget, ref cameraUpVector, out view);
Vector3.Up = <0,1,0>
```
Now lets define our world

- In Draw()

world = Matrix.Identity; // ?

Where is our world and what does it look like?
Defining Geometry

• XNA is right-handed?

• Make our geometry from triangles, why?
  • Always planar
  • Always convex
  • A few other rendering problems can’t happen.
private void InitializeVertices() {
    Vector3 p;  // position
    Vector2 t;  // textureCoordinates;
    vertices = new VertexPositionNormalTexture[3];
    p = new Vector3(-1, 1, 0);  // top left
    t = new Vector2(0, 0);
    vertices[0]=new VertexPositionNormalTexture(p,Vector3.Forward,t);
    p = new Vector3(1, -1, 0);  // bottom right
    t = new Vector2(1, 1);
    vertices[1]=new VertexPositionNormalTexture(p,Vector3.Forward,t);
    p = new Vector3(-1, -1, 0);  // bottom left
    t = new Vector2(0, 1);
    vertices[2]=new VertexPositionNormalTexture(p,Vector3.Forward,t);
}
Now to draw it

```csharp
BasicEffect effect = new BasicEffect(graphics.GraphicsDevice, null);
world = Matrix.Identity;
effect.World = world;
effect.Projection = projection;
effect.View = view;
effect.EnableDefaultLighting();
effect.Begin();
```
Now to draw it

```csharp
foreach (EffectPass pass in effect.CurrentTechnique.Passes)
{
    pass.Begin();
    graphics.GraphicsDevice.DrawUserPrimitives(
        PrimitiveType.TriangleList, vertices, 0, vertices.Length / 3);
    pass.End();
}
```

effect.End();
base.Draw(gameTime);

Effects allow us to get things on the screen. Actually, they are procedures that allow us to control the rendering of things. They use the HLSL which run on GPUs. More later, for now use built in BasicEffect

LineList
LineStrip
PointList
TriangleFan
TriangleList
TriangleStrip
Let’s add a texture

- Declare texture
  ```csharp
  private Texture2D texture
  ```
- Load it
  ```csharp
  LoadContent()
  Texture = content.Load<Texture2D>("texture");
  ```
- In Draw() after new BasicEffect & before effect.Begin()
  ```csharp
  Effect.TextureEnable = true;
  Effect.Texture = texture;
  ```
- <F5>
Triangle to Square

- We could create another triangle just like we did.
- But that would be 6 vertices, but how many does a square have?
- Instead let’s used indexed Vertex Buffers
private void InitializeVertices() {
    Vector3 p;  // position
    Vector2 t;  // textureCoordinates;
    vertices = new VertexPositionNormalTexture[4];
    p = new Vector3(-1, 1, 0);       //top left
    t = new Vector2(0, 0);
    vertices[0]=new VertexPositionNormalTexture(p,Vector3.Forward,t);
    p = new Vector3(1, -1, 0);       //bottom right
    t = new Vector2(1, 1);
    vertices[1]=new VertexPositionNormalTexture(p,Vector3.Forward,t);
    p = new Vector3(-1, -1, 0);      //bottom left
    t = new Vector2(0, 1);
    vertices[2]=new VertexPositionNormalTexture(p,Vector3.Forward,t);
    p = new Vector3(1, 1, 0);        //top right
    t = new Vector2(1, 0);
    vertices[3]=new VertexPositionNormalTexture(p,Vector3.Forward,t);
}
Texture Coordinates

- 2d
- (0,0) in upper left
- (1,1) in lower right
Private void InitializeIndices() {

indices = new short[6];

//triangle 1 (bottom portion)
indices[0] = 0; // top left
indices[1] = 1; // bottom right
indices[2] = 2; // bottom left

//triangle 2 (top portion)
indices[3] = 0; // top left
indices[4] = 3; // top right
indices[5] = 1; // bottom right

Notice order. It is important.
Changes to Draw()

```csharp
graphics.GraphicsDevice.DrawUserIndexedPrimitives(
    PrimitiveType.TriangleList, vertices, 0, vertices.Length,
    indices, 0, indices.Length / 3);
```

<F5>
Game Components

- We can use game components to separate out code
- Project->add new item->
  - Categories: XNA Game Studio 3.0
  - Templates: Game Component
  - Name: FPS.cs
- Change it to a DrawableGameComponent
Declarations

private float fps;
private float updateInterval = 1.0f;
private float timeSinceLastUpdate = 0.0f;
private float framecount = 0;
Constructor

```csharp
public FPS(Game game)
    : this(game, false, false, game.TargetElapsedTime) { }

public FPS(Game game, bool synchWithVerticalRetrace,
            bool isFixedTimeStep, TimeSpan targetElapsedTime) : base(game)
{
    GraphicsDeviceManager graphics =
        (GraphicsDeviceManager)Game.Services.GetService(
            typeof(IGraphicsDeviceManager));

    graphics.SynchronizeWithVerticalRetrace = synchWithVerticalRetrace;
    Game.IsFixedTimeStep = isFixedTimeStep;
    Game.TargetElapsedTime = targetElapsedTime;
}
```
Add it to our triangle demo

• Declare it
  private FPS fps;
• In Game1() constructor
  #if DEBUG
    fps = new FPS(this);
    Components.Add(fps);
  #endif
• Now you can use this to see how thing affect your frame rate.
• I get just under 600Hz on my laptop, got about 6000Hz before.
Now to draw it (revisit)

```csharp

BasicEffect effect = new BasicEffect(graphics.GraphicsDevice, null);
world = Matrix.Identity;
effect.World = world;
effect.Projection = projection;
effect.View = view;
effect.EnableDefaultLighting();
effect.Begin();
```

Why are we creating an effect every frame? It doesn’t change.

Let’s move this to Initialize, and make effect a class field.

5400Hz now

BasicEffect effect = new BasicEffect(graphics.GraphicsDevice, null);

world = Matrix.Identity;
effect.World = world;
effect.Projection = projection;
effect.View = view;
effect.EnableDefaultLighting();
effect.Begin();

Why are we creating a vertex declaration each frame?
Let’s move this to Initialize, and make effect a class field.

Still 5400Hz, no change on my laptop, but in the book they saw an improvement.
private void DrawRectangle(ref Matrix world) {
    effect.World = world;
    effect.Begin();
    foreach (EffectPass pass in effect.CurrentTechnique.Passes) {
        pass.Begin();
        graphics.GraphicsDevice.DrawUserIndexedPrimitives(
            PrimitiveType.TriangleList, vertices, 0,
            vertices.Length, indices, 0, indices.Length / 3);
        pass.End();
    }
    effect.End();
}
Modify Draw()

    // What we saw before
    world = Matrix.Identity;
    DrawRectangle(ref world);

    world = Matrix.CreateTranslation(new Vector3(3.0f, 0, -10.0f));
    DrawRectangle(ref world);
Can We Call DrawTriangle() again?

```csharp
world = Matrix.CreateTranslation(new Vector3(3.0f, 0, -10.0f));
DrawRectangle(ref world);
world = Matrix.CreateTranslation(new Vector3(-3.0f, 3.0f, -10.0f));
DrawRectangle(ref world);
```
Can We Call DrawTriangle() again?

world = Matrix.CreateTranslation(new Vector3(3.0f, 0, -10.0f));
DrawRectangle(ref world);
world = Matrix.CreateTranslation(new Vector3(-3.0f, 3.0f, -10.0f));
DrawRectangle(ref world);
world = Matrix.CreateRotationZ(angleX) * Matrix.CreateScale(.3f);
DrawRectangle(ref world);

Update()
angleX += .01f;

Notice that to compose transformations we multiply them together. Order matters, not commutative!
DrawUserIndexedPrimitives vs DrawIndexedPrimitives

- Make a copy of the solution folder
  - Rename the folder by adding DIP to the end
- Open the project
  - Open properties in solution explorer and click on AssemblyInfo.cs
    - Change the Title and the Guid
InitializeVertices()

vertexBuffer = new VertexBuffer(graphics.GraphicsDevice,
   VertexPositionNormalTexture.SizeInBytes * vertices.Length, ResourceUsage.WriteOnly,
   ResourceManagementMode.Automatic);
vertexBuffer.SetData(vertices);
//set the graphics device's vertex buffer
graphics.GraphicsDevice.Vertices[0].SetSource(
   vertexBuffer, 0,
   VertexPositionNormalTexture.SizeInBytes);

Create a VertexBuffer, then initialize the index buffer on the graphics device, tell it the size and to handle the memory automatically.