Software Engineering

- Look up Design Patterns
- Code Refactoring
- Code Documentation?

One of the lessons to learn for good design:

- Coupling: *the amount of interconnectedness between classes*
- Cohesion: *a measure of how closely related the responsibilities, data, and methods for a class are*

*For good code:* Increase cohesion and decrease coupling
Architecture: Code Review Session

Pair Review Guidelines
3D Graphics

Resources: outside Materials
Recommended Readings:
Chapters 5 (Tom Miller)
Chapter 11 (Aaron Reed)
Computer Graphics

- **Modeling**: math representation of a set of objects in a scene

- **Rendering**: drawing the images from model description to pixels on the display: combining lighting, objects, materials

- **Animation**: illusion of life
Some terms

- **Pixel**: short for Picture Element
- **Raster image**: image stored as an array of pixels usually with RGB values
- **Raster Display**: displays that show images as rectangular array of pixels
Computer Graphics

✦ **Modeling:** math representation of a set of objects in a scene

✦ **Rendering:** drawing the images from model description to pixels on the display: combining lighting, objects, materials

✦ **Animation:** illusion of life
Creating content

In 2D?
Pixel Art

Form of art whereby artists draw images by pixel using raster image software, e.g. paint or photoshop

Advantages:
- File types can be read by any program
- Easy for photo touch ups

Disadvantages:
- Cannot change the image too much
- Resizing problems
Vector Art

- Images represented as vectors (mathematically), lines, points, and curves

**Advantages:**
- Easy to change and manipulate objects
- Can convert to pixel or raster art
- Can resize

**Disadvantage:**
- Cannot use it for photos
- Not so good with a lot of color blends
2D Coordinate system
Creating Content

In 3D?
3D Model Representation

- Polygons and vertices
- Use triangles as simple primitive
3D Coordinate System

Or, if you are standing facing +ive X and your up is Y
Computer Graphics

- **Modeling**: math representation of a set of objects in a scene
- **Rendering**: drawing the images from model description to pixels on the display: combining lighting, objects, materials
- **Animation**: illusion of life
Displays

- Example raster display
- Frame Buffer which holds color value per pixel RGB and an alpha channel for translucency
Displays (LCD and LED)

* Both displays are based on independent 3 sub-pixels per pixel, showing RGB
Color

✦ Subtractive: subtracting color
   ✦ Paints

✦ Additive: is adding color
   ✦ Lights
   ✦ Displays
   ✦ CG
Color (Representation)

- 8-bit RGB fixed range: web, email apps
- 12-/14-bit RGB fixed range: photography
- 16-bit RGB fixed range: printing and image processing and photography
- 16-bit float RGB: HDR High Dynamic Range (real-time rendering)
- 32-bit float RGB: HDR
Projective Rendering

- Used by OpenGL & DirectX
- Efficient for real-time rendering
- For each object in the scene
  - Polyons are projected to 2D screen
  - Order/depth is conserved
  - Color is computed based on materials and light
Ray Tracing

- Shoot a ray from the eye to the scene
- Draw pixel based on color reflections from light sources and objects’ materials, etc.
- It is slow not usually real-time, but there are optimization techniques
Graphics Pipeline

Rendering Process
Model Transformation:

Take model from local coordinates to world coordinates
Illumination (Shading):
Color per vertex or pixel is calculated based on light sources, and object materials.
Viewing Transformation:

Color per vertex or pixel is calculated based on light sources, and object materials.
Clipping:
Removing parts of the 3D scene that is not visible
Projection to 2D space:

Projecting the 3D object to 2D space
Rasterization
Convert objects to pixels, interpolate color and depth
Rasterization

- Rasterization is the process by which 2D or 3D objects are converted into pixels
- Each pixel is represented in RBG, depth
Graphics Rendering Pipelines

Pre-1980’s Customized Software Rendering

Input Data
- Transformation and Lighting
- Primitive Setup
- Rasterization
- Pixel Processing
- Frame Buffer Blend

Pre 2001

Input Data
- Vertex Shading
- DX10 Geometry Shading
- Primitive Setup
- Rasterization
- Pixel Shading
- Frame Buffer Blend

DX8-DX10

Input Data
- Geometry Shading
- Primitive Setup
- Rasterization
- Pixel Shading
- Frame Buffer Blend

Larrabee

Alternative Larrabee: Customized Pipeline

Input Data
- Software Rendering

Frame Buffer

Frame Buffer

Frame Buffer Blend

Frame Buffer
Graphics Pipelines

3D Coordinate System

Or, if you are standing facing +ive X and your up is Y

It is important to know what coordinate system your artists used, otherwise the model will be flipped. To flip it back, multiply z coordinate by -1.
3D Graphics

Drawing a model in XNA
Chapters 5 (Tom Miller)
Get a Model

🔹 Turbo Squid
🔹 Or look for a model
🔹 Remember to credit the person when using it
Representation of a Model

✧ Model

✧ ModelMesh (e.g., head, body, hand, etc.)

✧ ModelMeshPart: smallest part sent to video card (e.g., triangle strips)

✧ Effect: describes how the object should be rendered (world, view, projection to describe how the object is positioned, lighting, etc.)
Effect Classes in XNA

- **BasicEffect** (includes basic lighting: `BasicEffect.EnableDefaultLighting()`)
- **Effect** (use if you want to create your own shaders)
Adding a model in XNA

```csharp
//model
Model model;

Vector3 modelPosition;
Vector3 cameraPosition;
float modelRotation;
Vector3 cameraForward;

int toggle = 0;

public Game1()
{
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";
}
```
/// and initialize them as well.
/// </summary>
protected override void Initialize()
{
    modelPosition = Vector3.Zero;
cameraPosition = new Vector3(0, 20, -50);
cameraForward = new Vector3(0, 0, -1);
modelRotation = 0.0f;

    base.Initialize();
}

/// <summary>
/// LoadContent will be called once per game and is the place to load
/// all of your content.
/// </summary>
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);
    model = Content.Load<Model>("warlock");
}
Adding a model in XNA

```csharp
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // adding drawing for the model
    Matrix[] transforms = new Matrix[model.Bones.Count];
    model.CopyAbsoluteBoneTransformsTo(transforms);
```
// adding drawing for the model
Matrix[] transforms = new Matrix[model.Bones.Count];
model.CopyAbsoluteBoneTransformsTo(transforms);

// Draw the model. A model can have multiple meshes, so loop.
foreach (ModelMesh mesh in model.Meshes)
{
    // This is where the mesh orientation is set, as well
    // as our camera and projection.
    foreach (BasicEffect effect in mesh.Effects)
    {
        effect.EnableDefaultLighting();

        effect.World = transforms[mesh.ParentBone.Index] *
            Matrix.CreateRotationY(modelRotation) *
            Matrix.CreateTranslation(modelPosition);

        effect.View = Matrix.CreateLookAt(cameraPosition,

        effect.Projection = Matrix.CreatePerspectiveFieldOfView(
            1.0f, 1000.0f);
    }

    // Draw the mesh, using the effects set above.
    mesh.Draw();
}

def Draw(gameTime)
Matrix in XNA

- **World Matrix**: where the model is in relation to the world
  - \( \text{WorldMatrix} = \text{mesh} * \text{scale} * \text{rotate} * \text{translation matrix} \);

- **View Matrix**: where the camera sits
  - \( \text{CreateLookAt} \) (\( \text{cameraPos} \), \( \text{cameraTarget} \), \( \text{cameraUpVector} \));

- **Projection**: how 3D space is projected into 2D screen
  - **CreatePerspectiveFieldOfView** (\( \text{fieldOfView} \), \( \text{aspectRatio} \), \( \text{nearPlane} \), \( \text{farPlane} \))
  - FieldOfView most often = MathHelper.PiOver4
Class Exercise

Get a Model and add it in a 3D Scene in XNA
Moving the Model around

Vector Math
Outline

- Scaling (generalized from 2D)
- Rotation (rotation about a line and not points)
- Translation
Vector Transformation: Scaling

Vector Transformation:

In 2D

\[
\begin{pmatrix}
  s_x & 0 \\
  0 & s_y
\end{pmatrix}
\begin{pmatrix}
  x \\
  y
\end{pmatrix}
\]

In 3D

\[
\begin{pmatrix}
  s_x & 0 & 0 \\
  0 & s_y & 0 \\
  0 & 0 & s_z
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix}
\]
Vector Transformation: Rotation

- **Around origin**
  
  In 2D
  
  \[
  \begin{bmatrix}
  \cos \theta & -\sin \theta \\
  \sin \theta & \cos \theta
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

  In 3D
  
  Transformation Matrix

- **x-axis**
  
  \[
  \begin{bmatrix}
  1 & 0 & 0 \\
  0 & \cos \theta & -\sin \theta \\
  0 & \sin \theta & \cos \theta
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y \\
  z
  \end{bmatrix}
  \]

- **y-axis**
  
  \[
  \begin{bmatrix}
  \cos \theta & 0 & \sin \theta \\
  0 & 1 & 0 \\
  -\sin \theta & 0 & \cos \theta
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y \\
  z
  \end{bmatrix}
  \]
Rotations

- Orientation: yaw, pitch and roll
  - Yaw: around the Y
  - Pitch: around the Z
  - Roll: around X
2D Translation of an object

• Represent the point \( x, y \) by vector

\[
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]

• Thus:

\[
\begin{pmatrix}
  x' \\
  y' \\
  1
\end{pmatrix} =
\begin{bmatrix}
  1 & 0 & x_t \\
  0 & 1 & y_t \\
  0 & 0 & 1
\end{bmatrix}
\begin{pmatrix}
  x \\
  y \\
  1
\end{pmatrix}
= \begin{bmatrix}
  x + x_t \\
  y + y_t \\
  1
\end{bmatrix}
\]

Affine Transformation
3D Translation of an object

- Represent the point x, y by vector

Thus:

\[
\begin{bmatrix}
  x \\
y \\
z \\
1
\end{bmatrix}
\]

Homogenous Coordinates

Affine Transformation

\[
\begin{pmatrix}
x' \\
y' \\
z' \\
1
\end{pmatrix} =
\begin{bmatrix}
1 & 0 & 0 & x_t \\
0 & 1 & 0 & y_t \\
0 & 0 & 1 & z_t \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix} =
\begin{bmatrix}
x + x_t \\
y + y_t \\
z + z_t \\
1
\end{bmatrix}
\]
Vector Transformation: Translation

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
T.x & T.y & T.z & 1 & 0
\end{bmatrix}
\]
Rotations (Affine Transformation)

\[
X = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos(x) & -\sin(x) & 0 \\
0 & \sin(x) & \cos(x) & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
Y = \begin{bmatrix}
\cos(y) & 0 & \sin(y) & 0 \\
0 & 1 & 0 & 0 \\
-\sin(y) & 0 & \cos(y) & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
Z = \begin{bmatrix}
\cos(z) & -\sin(z) & 0 & 0 \\
\sin(z) & \cos(z) & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Scaling (Affine Transformation)

Scaling

\[
\begin{bmatrix}
\text{scale} & 0 & 0 & 0 \\
0 & \text{scale} & 0 & 0 \\
0 & 0 & \text{scale} & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

When you are doing transformation on object, order is important
Vector Math: Dot Product

- Is a scalar
- Normalize vectors before use, formula is:
  
  \[
  \text{float dotproduct} = (v1.x \times v2.x) + (v1.y \times v2.y) + (v1.z \times v2.z)
  \]

- Used to calculate angle between two vectors

\[
\theta = \cos^{-1}\left(\frac{a \cdot b}{|a||b|}\right)
\]
Dot product:
- Same vector?
- Opposite vector?
- Orthogonal vector?

Why is the Dot product important?
Vector Math: Cross Product

✧ Is a vector

✧ Normalize vectors before use, formula is:

```cpp
Vector3 crossproduct;
crossproduct.x = (v1.y*v2.z) - (v2.y*v1.z);
crossproduct.y = (v1.z*v2.x) - (v2.z*v1.x);
crossproduct.z = (v1.x*v2.y) - (v2.x*v1.y);
```

✧ Used to calculate normal vector between two vectors
Vector Math

Unit vector:

- vector of length of 1.0
- To unitize or normalize your vector:

```csharp
Vector3 destination = new Vector3(10, 15, 10);
Vector3 origin = new Vector3(2, 15, 0);
Vector3 direction = destination - origin;
direction.Normalize();
```
Vector Transformation: Scaling

Vector Transformation:

In 2D

\[
\begin{pmatrix}
s_x & 0 \\
0 & s_y
\end{pmatrix}
\begin{pmatrix}
x \\
y
\end{pmatrix}
\]

In 3D

\[
\begin{pmatrix}
s_x & 0 & 0 \\
0 & s_y & 0 \\
0 & 0 & s_z
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z
\end{pmatrix}
\]

In XNA:
Create a scale variable Vector3, then Matrix.CreateScale (scale)
E.g., Rotate towards object

```csharp
Vector3 destination = new Vector3(10, 15, 10);
Vector3 origin = new Vector3(2, 15, 0);
Vector3 direction = destination - origin;
Vector3 facingDirection = new Vector3(0, 0, 10);
direction.Normalize();
facingDirection.Normalize();

//getting the rotation angle
float rotationAngle;
rotationAngle = Math.Acos(Vector3.Dot(direction, facingDirection));
```
E.g., Rotate towards object

```csharp
Vector3 destination = new Vector3(10, 15, 10);
Vector3 origin = new Vector3(2, 15, 0);
Vector3 direction = destination - origin;
Vector3 facingDirection = new Vector3(0, 0, 10);
direction.Normalize();
facingDirection.Normalize();

//getting the rotation angle
float rotationAngle;
rotationAngle = Math.Acos(Vector3.Dot(direction, facingDirection));
```

Problem, don’t know which direction to rotate, need Cross Product
Matrix Class in XNA

- Matrix.CreateTranslation(translationVector3);
- Matrix.CreateRotationX(angle);
- Matrix.CreateRotationY(angle);
- Matrix.CreateRotationZ(angle);
- Matrix.CreateScale(scale);
Matrix Class in XNA

- \( \text{worldMatrix} = \text{scale} \times \text{rotate} \times \text{translate}; \)
Class Assignment

Try moving the model forward and backward
Assignment # 2

Online