OpenTK

Richard B. Johnson
Nick Van Aartsen

CMSC498M - Game Programming
University of Maryland, College Park
What is OpenTK?

- Provides
  - Low level access to OpenGL
  - Audio APIs for sound and music
  - Game Window environment
  - Keyboard and mouse input
  - Math library

- Developed exclusively for the C# programming language
Why C# and not C++ or Java?

- Fast
  - Compiled language: No JIT compiling
  - Faster to develop in than C++

- User friendly
  - Object oriented + added features
    - C++ = (C)++
    - C# = (C++)++
  - Built in memory management: no more leaks
  - No more need for header files!
  - Easier to read

- Cross-platform distribution with Mono
using OpenTK;
using OpenTK.Graphics.OpenGL;

class Game : OpenTK.GameWindow
{
    public Game()
    public override void OnLoad(EventArgs e)
    protected override void OnResize(EventArgs e)
    public override void OnUpdateFrame(FrameEventArgs e)
    public override void OnRenderFrame(FrameEventArgs e)
    static void Main(string[] args)
}
Game Window - Constructor

```csharp
public Game()
    : base(1024, 768, GraphicsMode.Default, "Title")

public Game()
    : base(
        DisplayDevice.Default.Width,
        DisplayDevice.Default.Height,
        GraphicsMode.Default,
        "Title",
        GameWindowFlags.Fullscreen,
        DisplayDevice.Default)
```
public override void OnLoad(EventArgs e)
{
    // Game and scene setup calls
    ...

    // Render initialization calls
    GL.ClearColor(System.Drawing.Color.Black);
    GL.Enable(EnableCap.DepthTest);
    ...
}
Game Window - OnResize

```csharp
protected override void OnResize(ResizeEventArgs e)
{
    GL.Viewport(0, 0, Width, Height);
    GLMatrixMode(MatrixMode.Projection);
    GLLoadIdentity();

    // Set perspective mode
    Glu.Perspective(45.0, Width / (double)Height, 1.0, 64.0);
}
```
Game Window - OnResize

protected override void OnResize(ResizeEventArgs e)
{
    GL.Viewport(0, 0, Width, Height);

    SetPerspectiveMode(45.0, Width / (double)Height, 1.0, 64.0);

    // DEPRECATED!!!
    Glu.Perspective(45.0, Width / (double)Height, 1.0, 64.0);
}

protected static void SetPerspectiveMode(
    double fov, double aspect, double near, double far)
{
    double y = Math.Tan(Scalar.ToRadians(fov) / 2.0) * near;
    double x = y * aspect;

    GL.MatrixMode(MatrixMode.Projection);
    GL.LoadIdentity();
    GL.Frustum(-x, x, -y, y, near, far);
}
protected static void SetOrthogonalMode(
    double scale, double aspect, double near, double far)
{
    double y = scale * near / 2;
    double x = y * aspect;

    GL.MatrixMode(MatrixMode.Projection);
    GL.LoadIdentity();
    GL.Ortho(-x, x, -y, y, near, far);
}
Game Window - OnUpdateFrame

```csharp
public override void OnUpdateFrame(UpdateFrameEventArgs e)
{
    if (Keyboard[Key.Escape])
        Exit();

    ...

    Process input and AI
    Process received packets
    Physics and Collision
    Update game entities
    Send outgoing network packets
}
```
public override void OnRenderFrame(RenderFrameEventArgs e) {
    GL.Clear(
        ClearBufferMask.ColorBufferBit |
        ClearBufferMask.DepthBufferBit);
    ...
    // Render scene and game objects here:
    ...
    SwapBuffers();
}
Game Window - Main

[STAThread]
static void Main()
{
    using (Game myGame = new Game())
    {
        myGame.Run(30.0, 60.0);
    }
}

Frames per Second

Updates per Second
OpenTK Input

using OpenTK;
using OpenTK.Input;

- Support exists for:
  - Keyboard
  - Mouse
  - Gamepad
  - Joystick
Input – Event Handlers

{ 
...
Mouse.Move += HandleMouseMove;
Mouse.ButtonDown += HandleMouseDown;
Mouse.ButtonUp += HandleMouseUp;
}

void HandleMouseDown(
    object sender, MouseButtonEventArgs m)
{
    ...
}
using OpenTK;

- Support exists for:
  - Vector
  - Matrix
  - Quaternion
  - Bezier Curves
Shortcomings of OpenTK 1.0

- Fairly new / Refactoring between versions
- Deprecated or no support for GLU and GLUT
  - No more GLU.LookAt()
- How to get around this?
  - Either use GL.Translate() and GL.Rotate()
  - Or calculate your own and use GL.MultMatrix()
Our Solution
But first some OpenGL facts

- Right-handed coordinate system
  - Looking down negative z-axis
- Column major matrices
  - Multiplication in reverse order
- Memory storage and access:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>7</td>
<td>11</td>
<td>15</td>
</tr>
</tbody>
</table>
Object Representation

- Every object is represented in 3D space as a:
  - Point for position (P)
  - Local axis unit vectors:
    - Vector for look at direction (L)
    - Vector for up direction (U)
    - Vector for right (R)
  
  \[ R = \text{crossProduct}(L, U) \]
## Transformation Matrices

### Translation

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>P_x</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>P_y</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>P_z</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

### Translation Inverse

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-P_x</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-P_y</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-P_z</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

### Rotation

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_i</td>
<td>U_i</td>
<td>-L_i</td>
<td>0</td>
</tr>
<tr>
<td>R_j</td>
<td>U_j</td>
<td>-L_j</td>
<td>0</td>
</tr>
<tr>
<td>R_k</td>
<td>U_k</td>
<td>-L_k</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### Rotation Inverse

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_i</td>
<td>R_j</td>
<td>R_k</td>
<td>0</td>
</tr>
<tr>
<td>U_i</td>
<td>U_j</td>
<td>U_k</td>
<td>0</td>
</tr>
<tr>
<td>-L_i</td>
<td>-L_j</td>
<td>-L_k</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
## Implementation

```
// Apply camera inverse transformation
GLMatrixMode(MatrixMode.Modelview);
GLLoadIdentity();
GLMultMatrix(camera.Control.InverseRotation);
GLMultMatrix(camera.Control.InverseTranslation);

// Apply object forward transformation
GLPushMatrix();
GLMultMatrix(object.Control.Translation);
GLMultMatrix(object.Control.Rotation);

GLPopMatrix();

GLPopMatrix();
```

\[
R_c^{-1} T_c^{-1} (T_o R_o \text{ object})
\]
Any Questions?
Smoothing the Experience

- Smoothing improves the user interaction
  - aka “makes things feel really good”
- Closes the gap of professionalism
  - Even FPSes smooth movement
- Can, and should be subtle
  - Do not slow the game down
  - Do provide an organic, continuous motion
Smoothing - General

- Monitor **at least** twice the points
  - Minimally - position and destination
  - Likely - velocity/acceleration vectors
- deltaTime – everything depends on this
  - Only move as much as time has passed
- Apply smoothing function to approach dest.
  - Spring Force
  - Acceleration/Deceleration
  - Bezier, sinusoidal curves, etc
**Spring Force**

- Useful for camera movement, fps movement
  - Moves *quickly* – slows near the end
- Hooke’s Law
  - $F = -kx$
  - $k = \text{constant}, \ x = \text{displacement from rest}$
- Dampening factor
  - Prevents high speeds, minimizes “swinging”
  - $F = -cv$
  - $c = \text{constant}, \ v = \text{velocity of object}$
Spring Force

- Based on springConst and dampConst

```java
Vector displace = src.Difference(dst);
Vector velocity = prv.Difference(src) * deltaTime;
double dispMag = displace.Magnitude();

double forceMag =
    (springConst * (springLen - dispMag)) +
    (dampConst * ((displace * velocity) / dispMag));

displace = displace.Normal() * (forceMag * deltaTime);

src.Add(displace);
```
Spring Force - Shortcut

- Too much math for a simple function?
  - Just let me program the game!

- Okay, you can approximate this

```csharp
Vector displace = src.Difference(dst);
//slowConst is generally from 0.5 to 2.0 in range
double dist = displace.Magnitude() * slowConst * deltaTime;
Vector move = displace.Normal() * dist;
src.Add(move);
```

- Moves a set % closer each pass (roughly)
  - i.e. 1.0 -> 0.5 -> 0.25 -> 0.125... etc

- But, needs some boundary checks
Accelerating to a Point

- Often you want to approach a point given constraints
  - Max speed, max acceleration
  - Decelerate at a fixed rate, stop on the target.
- We need it to be dynamic
  - What if the target changes?
- Spring is not really appropriate
  - Acceleration would be impacted by distance
  - Too many constraints needed on spring formula
Accelerating to a Point

- Solution is “simple”
  - Each object has an accel. and decel. speed
  - Each frame, calculate if we should **speed up** or **slow down**
    - If so, accelerate, if not, decelerate
  - Bound speed by \([0, \text{max}_\text{speed}]\)
- Seems expensive, but is not too bad
- Easy to manage – minimal persistant data
- Adjusts easily to new parameters
Accelerating to a Point

- How do we know to accelerate or decelerate?
  - If we started decelerating now, would we reach the target before we stopped?
- A couple of quick steps
  - $t = \frac{\text{speed}}{\text{decelRate}}$ //time required to stop
  - $\text{dist} = 0.5 \times \text{decelRate} \times t^2 + \text{speed} \times t$
- If src + dist > dst, we can safely decelerate
- Otherwise, full speed ahead! (accelerate)
  - Limited by speed cap, so this is controlled
Other notes

- All of these ideas can be easily implemented in 1, 2, or 3 (or 4!) dimensions
  - Hurray vector math!
- Smoothing of rotational vectors is often also desirable
  - Just as before, set a target and angular velocity
- Again, can improve professional appearance
  - Or, make your controls lousy and slow. Careful.