Animation and Rendering Basics

• The game loop
• Simple physics
• Euler steps
• Collision detection
• Collision response

Examples are in C#
The game loop

- Core component of most animation applications

```plaintext
while (not done)
    Process any input
    Advance
    Render
    Play sounds
    Message pump
end while
```
Example

- Simple C# example:

```csharp
while (this.Created) {
    Advance();
    Render();
    Application.DoEvents();
}
```
private void Render()
{
    if (device == null)
        return;

    device.Clear(ClearFlags.Target, System.Drawing.Color.Black, 1.0f, 0);

    device.RenderState.ZBufferEnable = false; // We'll not use this feature
    device.RenderState.Lighting = false;      // Or this one...
    device.RenderState.CullMode = Cull.None;   // Or this one...

    device.Transform.Projection = Matrix.OrthoOffCenterLH(0, worldW, 0, worldH, 0, 1);

    //Begin the scene
    device.BeginScene();

    foreach (Polygon p in objects)
    {
        p.Draw(device);
    }

    //End the scene
    device.EndScene();
    device.Present();
}
Drawing a Square

/// <summary>
/// Current player position
/// </summary>
private Vector2 position = new Vector2(0, 0);

DrawQuad(new Vector2(0, 1) + position,
          new Vector2(0, 2) + position,
          new Vector2(1, 2) + position,
          new Vector2(1, 1) + position, Color.FromArgb(255, 255, 0));
private VertexBuffer drawQuadV = null;

private void DrawQuad(Vector2 v1, Vector2 v2, Vector2 v3, Vector2 v4, Color c)
{
    if (drawQuadV == null)
    {
        drawQuadV = new VertexBuffer(typeof(CustomVertex.PositionColored), // Type of vertex
                                      4, // How many
device, // What device
                                      0, // No special usage
                                      CustomVertex.PositionColored.Format,
                                      Pool.Managed);
    }

    GraphicsStream gs = drawQuadV.Lock(0, 0, 0); // Lock the background vertex list
    int clr = c.ToArgb();

    gs.Write(new CustomVertex.PositionColored(v1.X, v1.Y, 0, clr));
    gs.Write(new CustomVertex.PositionColored(v2.X, v2.Y, 0, clr));

    drawQuadV.Unlock();

    device.SetStreamSource(0, drawQuadV, 0);
    device.VertexFormat = CustomVertex.PositionColored.Format;
    device.DrawPrimitives(PrimitiveType.TriangleFan, 0, 2);
}
A Box class

/// <summary>
/// A box to draw
/// </summary>
private Box box = new Box();

box.Draw(device, 2, 2, 1, 1, Color.Green);
public class Box
{
    private VertexBuffer vertices = null;

    public void Draw(Device device, float x, float y, float wid, float hit, System.Drawing.Color color)
    {
        if (vertices == null)
        {
            vertices = new VertexBuffer(typeof(CustomVertex.PositionColored),
                4, // How many
                device, // What device
                0, // No special usage
                CustomVertex.PositionColored.Format,
                Pool.Managed);
        }

        GraphicsStream gs = vertices.Lock(0, 0, 0); // Lock the background vertex list
        int clr = color.ToArgb();

        gs.Write(new CustomVertex.PositionColored(x - wid/2, y - hit/2, 0, clr));
        gs.Write(new CustomVertex.PositionColored(x - wid/2, y + hit/2, 0, clr));
        gs.Write(new CustomVertex.PositionColored(x + wid/2, y + hit/2, 0, clr));
        gs.Write(new CustomVertex.PositionColored(x + wid/2, y - hit/2, 0, clr));

        vertices.Unlock();
        device.SetStreamSource(0, vertices, 0);
        device.VertexFormat = CustomVertex.PositionColored.Format;
        device.DrawPrimitives(PrimitiveType.TriangleFan, 0, 2);
    }
}
Advance

• Move things
• Resolve any collisions
  – What’s a collision?
Simple 2D Physics

• Motion of objects in 2D
  – May or may not obey the laws of physics
  – We’ll create general ideas

  – Most of this can be extended to 3D pretty easily, except rotation is more complicated
  – This is a basic introduction and omits details of objects in continuous contact
The basics

• Let $p(t)$ be the position in time.
  – We’ll drop the $(t)$ and just say $p$

• Other values:
  – $v$ – velocity
  – $a$ - acceleration

\[ v = \frac{dp}{dt} \]
\[ a = \frac{dv}{dt} = \frac{d^2 p}{dt^2} \]

Velocity is the derivative of position

Acceleration is the derivative of velocity
Vector calculus

• Really, p(t) is a double, right?
  – Sometimes
  – Think of these equations as two equations, one for each dimension
  – This is vector calculus

\[
\begin{bmatrix}
  v_x(t) \\
  v_y(t)
\end{bmatrix} = \begin{bmatrix}
  \frac{dp_x}{dt} \\
  \frac{dp_y}{dt}
\end{bmatrix}
\]
What about real objects?

• Some of the things we’ll know about a real object:
  – Position vector (p)
  – Velocity vector (v)
  – Acceleration vector (a)

• We might also know:
  – Mass

• More to come...
Variables to keep track of state

```csharp
private Vector2 p = new Vector2(0, 0); // Location
private Vector2 v = new Vector2(0, 0); // Linear velocity
private Vector2 a = new Vector2(0, 0); // Linear acceleration
```
Example: Air Resistance

- The resistance of air is proportional to the velocity
  - \( F = -kv \)

- We know \( F = ma \), so: \( ma = -kv \)

\[
ma = -kv
\]

\[
m \frac{dv}{dt} = -kv
\]

So, how can we solve?
Euler’s method

\[ m \frac{dv}{dt} = -kv \]

\[ \frac{dv}{dt} = - \frac{kv}{m} \]

\[ \Delta v \approx - \frac{kv}{m} \Delta t \]

\[ v_{i+1} \approx v_i - \frac{kv}{m} \Delta t \]
public void Step(float dt) {
    v.X += a.X * dt;
    v.Y += a.Y * dt;
    p.X += v.X * dt;
    p.Y += v.Y * dt;
    //…
}

- Velocity update
- Position update
Another version

```java
public void Step(float dt) {
    playerSpeed += playerAccel * dt;
    playerLoc += playerSpeed * dt;
}
```

Velocity update

Position update
What about orientation?

• Let $\Omega$ be the orientation (angle, radians)
  – Easy in 2D, not so easy in 3D
  – We’ll sometimes call this $r$

• **How fast we are spinning** is the rotational velocity $\omega$ (radians per second)
  – We’ll sometimes call this $rv$ or $r_v$

• We talk about **rotational** and **linear** terms
Rotational vs. Linear

• Linear
  – Position, velocity, acceleration

• Rotational
  – Orientation, rotational velocity, rotational acceleration
A more complete state as a class

class State
{
    public Vector2 p = new Vector2(0, 0); // Location
    public Vector2 v = new Vector2(0, 0); // Linear velocity
    public Vector2 a = new Vector2(0, 0); // Linear acceleration

    public float r = 0; // Angle
    public float rv = 0; // Rotational velocity
}

Note: Rotation is 1D
Complete Euler step

```java
public void Step(float dt)
{
    state.v.X += state.a.X * dt;
    state.v.Y += state.a.Y * dt;
    state.p.X += state.v.X * dt;
    state.p.Y += state.v.Y * dt;
    state.r += state.rv * dt;

    Move();
}
```

Rotational velocity update

Creates a set of points moved into place
Coding Examples

• Creating state
• Euler steps
• Arrow keys
• Jumping
Defining a polygon

• Rules + Guidelines
  – Convex polygons only
    • You can wrap a rubber band around it and it all of the faces
    • Any line segment connecting two points inside the polygon does not exit the polygon
  – Define the polygon in some “base pose”
    • Put the center of gravity at (0,0)
  – Supply the vertices in clockwise order!
Polygons

Not convex!

Wrong order!
Transforming the polygon vertices

• To draw, we convert the vertices to the destination location
  – Source to destination transformation
  – Rotate, then translate

\[
x' = x \cos(a) - y \sin(a) + p_x
\]
\[
y' = x \sin(a) + y \cos(a) + p_y
\]
private List<Vector2> verticesB = new List<Vector2>();  // Before transformation
private List<Vector2> verticesM = new List<Vector2>();  // After transformation

// Add a vertex to the polygon. Must be called before the
// first rendering of the polygon.
void AddVertex(Vector2 vertex)  
  { verticesB.Add(vertex);  }

// Create the array of the verticies after being moved
// Converts verticesB to verticesM.
private void Move()
{
  verticesM.Clear();              // Destination
  float ca = (float)Math.Cos(state.r);
  float sa = (float)Math.Sin(state.r);

  foreach (Vector2 v in verticesB)
  {
    Vector2 vp = new Vector2(ca * v.X - sa * v.Y + state.p.X,
                             sa * v.X + ca * v.Y + state.p.Y);

    verticesM.Add(vp);
  }
}
Is this enough?

• When do you need how much physics?
  – Up till now can simulate
    • Continuous motion
    • Jumping
      – Set initial velocity, gravity to decelerate
    • Air resistance or friction
      – $a = -k_p v$, $r_a = -k_r v$
    • Gravity
Forces

• What if some force is applied to our objects?
  – We need to know characteristics of the objects
    • Mass
      – Resistance to linear motion
    • Moment of inertia
      – Resistance to rotary motion
Force and torque

• $F_i(t)$
  – Force on point $i$ at time $t$
  – Vector, of course

Torque on point $i$: $\tau_i(t) = (p_i(t) - p(t))_\perp \cdot F_i(t)$

Total Force: $F(t) = \sum F_i(t)$

Total Torque: $\tau(t) = \sum \tau_i(t) = \sum (p_i(t) - p(t))_\perp \cdot F_i(t)$
The “perp” operator

- It’s a 90 degree counterclockwise rotation of a vector. \[ \tau_i(t) = (p_i(t) - p(t))_\perp \cdot F_i(t) \]

Torque on point i

\[(x, y)_\perp = (-y, x)\]
The “perp” operator

- It’s a 90 degree counterclockwise rotation of a vector. 

\[ \tau_i(t) = (p_i(t) - p(t))_\perp \cdot F_i(t) \]

Torque on point i

\[ (x, y)_\perp = (-y, x) \]

Vector2 \( r = \text{new Vector2}(p_i.X - p.X, p_i.Y - p.Y); \)
float torque = -r.Y * F.X + r.X * F.Y;

The “\( r \)” vector: 

\[ r_i(t) = p_i(t) - p(t) \]
Compute total forces

Total Force: \[ F(t) = \sum F_i(t) \]

Total Torque: \[ \tau(t) = \sum \tau_i(t) = \sum \left( p_i(t) - p(t) \right)_\perp \cdot F_i(t) \]
Moment of inertia

\[ I = \int m_p \left| r_p \right|^2 dp \]

- \( m_p \): mass at a point
- \( r_p \): vector from center of mass to the point

What are the characteristics?

What does large vs. small mean?

How to we get this value?
Torque and force

\[ a_r = \frac{\tau}{I} \]

\[ a_l = \frac{F}{M} \]

Rotational Acceleration

Linear Acceleration
Simple dynamics

• Calculate/define center of mass (CM) and moment of inertia (I)
• Set initial position, orientation, linear, and angular velocities
• Determine all forces on the body
• Linear acceleration is sum of forces divided by mass
• Angular acceleration is sum of torques divided by I
• Numerically integrate (Euler step) to update position, orientation, and velocities