CS3505/5020
Software Practice II

XNA overview
Representations in Simulations and Games
Homework Q/A
XNA – *(XNA Not Acronymed 😊)*

- Derive from Game class
- **Must override:**
  - Initialize – setup (one time)
  - Update – compute game logic
  - Draw – display
- **Separation of logic and display is an important model**
  - Not just for games, but in general this is a good idea
- **Game.Run starts the game loop**
**Fixed-Step Game Loop**

- Set `Game.IsFixedTimeStep` to true (default)
- `TargetElapsedTime` – default 1/60\(^{th}\) of a sec
- Slow – `IsRunningSlowly` (bool)
Variable-Step Game Loop

- Set `Game.IsFixedTimeStep` to false
- `TargetElapsedGameTime` ignored
- `ElapsedGameTime` – time since last call to `Update`
  - This works in either model
Contrast The Two Models

- What if user has lots of other tasks running?
- What if you use two different computers?
- If you have a sprite moving across the screen, how will it behave?
- Note – debugger entrance causes timer to be suspended
**GameComponent**

- Game provides a collection of GameComponent objects or DrawableGameComponent objects
- Mechanism to modularize your solution
  - Good if you want to create components that are reusable
- Game.Components.Add
Game Content

● Problem – how do you make it easy for content authors and programmers to work together?
  – Artists create content using many different Digital Content Creation (DCC) tools

● Problem – how do you deal with Windows and Xbox 360 assets?

● Solution: XNA Content Pipeline
Content Pipeline Architecture

- XNA provides content importers and processors that will load and process the content for your game
  - Creates Managed Object with Strong Typing
  - Content manager (Content.Load…)

![Content Pipeline Diagram](image-url)
Standard Importers

- Autodesk FBX format
- Directx Effects File format
- Sprite Font file
- Texture importer (\.bmp, .dds, .dib, .hdr, .jpg, .pfm, .png, .ppm, and .tga)
- DirectX X file format (coordinates)
- XACT for sounds
- XML Content
- Note – 3rd party importers available or you can write your own
Understanding the Displays

Windows Desktop

(X, Y)

ClientBounds.Width

ClientBounds.Height

DisplayMode.Width

DisplayMode.Height

Xbox 360 Display

Score: 10000

ClientBounds.Width

DisplayMode.Width

ClientBounds.Height

DisplayMode.Height
Back Buffer

- Title safe area is inner 80% of the back buffer
2D Graphics

- Drawing sprites
- Sprite origin normally upper left corner, but you can change that
- Sprite depth (floating point number) between 0 (front) and 1 (back)
- Use sourceRectangle if you want to draw part of a texture
- You can also scale the texture in the Draw command (uniform or non-uniform)
- SpriteBatch lets you do a transformation matrix (rotation, translation, scaling)
How to structure a robust simulation or game

- Strongly decouple the simulation from the drawing.
  - The simulation should not depend on screen sizes, pixels, or images (see following slides).
  - Keep a distinct game state that is advanced in small, deterministic steps.
    » Without external input, a game in state A should always advance to state B.
  - Ensure that the drawing mechanism only pulls data from a single, static game state.
Representing coordinates

- Easiest method – objects exist in screen space.
  - An object’s coordinates are mapped directly to screen coordinates. This shape is at (17, 5):
Representing coordinates

- Easiest method – objects exist in screen space.
  - An object’s coordinates are mapped directly to screen coordinates
  - Disadvantages:
    » Screen coordinates not fixed
    » Aspect ratios not fixed
    » Screen coordinates are integers, fractional values needed for motion
    » Difficult to apply rotations or scaling
Representing coordinates

- Better method – objects exist in world coordinate space.
  - An object’s coordinates are in an arbitrary coordinate system. This shape is at (46.2, 21.8):
Representing coordinates

- A transform is needed to convert this to screen coordinates.
  - Assume ALL values and sizes are in world space
Representing coordinates

\[ \text{pixelX} = \frac{(\text{ShapeX} - \text{ScreenX})}{\text{screenWidth}} \times \text{pixelWidth} \]

\[ \text{pixelY is similar, must account for flipped y} \]
Representing coordinates

- Additional coordinate systems commonly used: World space, Object space, View space, etc.
- Decoupling of coordinate spaces insulates simulation objects from the way they are viewed (or used).
Representing coordinates

- Coordinate systems do not have to be at right angles to each other:
  - This is commonly how rotations are represented
Representing coordinates

- Coordinate systems do not have to be at right angles to each other:
  - Conversion from/to a system requires a translation (movement of the origin) and a rotation, and then a reverse translation.
  - Graphics libraries almost always have the notion of a transformation matrix that encapsulates these operations in a simple linear algebra form. (No trig required.)
Representing motion

- If you know where the object is, and you know where you want it to be, it is easy enough to compute a displacement vector:
  - $V = (\Delta x, \Delta y)$

- If you want this motion to take 3 seconds, then move 1/3 this distance each second
  - Divide by desired elapsed time
  - $V = (\Delta x/time, \Delta y/time)$
Representing motion

- The resulting vector is your velocity vector
  - Two components, velocityX and velocityY
  - Units are distance/time
- To simulate motion, simply add the velocity*timestep to your position several times:
- How fast are you moving?
  - Find mag. of vector using Pythagorean theorem
Acceleration

- Velocity represents a change in position over time.
- Acceleration represents a change in velocity over time.
  - Represent is as a vector: \( A = (\Delta vx, \Delta vy) \)
  - Every time step, add the acceleration to the velocity. The velocity will change proportionally.
  - Some accelerations (like gravity) seem constant at long distances, but can vary greatly \((r^2)\) at some scales. Be careful to simulate correctly when appropriate.
Acceleration

- Typical accelerations include:
  - Internal motivation of the object, i.e. gas pedal
  - Gravity
  - Wind resistance / surface resistance / drag
  - Other fields (magnetism, etc.)

- If object mass is important, it is better to model: \( \text{acceleration} = \frac{\text{force}}{\text{mass}} \)

- For games, visually appropriate values often supersede exact physics.
What about changes to acceleration?
What about changes to acceleration?

- Important to ‘smoothness’ feel of a simulation.
- Humans notice changes in the second derivative of velocity.
- Official name: Jerk*
- Don’t need to simulate it, but you might want to make sure your simulation is continuous in the second derivative of velocity.

* Not verified, but web vetted.
Doing proper collisions is extremely tough.
- Shapes often have irregular borders.
- Collisions can depend on graphic renderings, breaking the separation of the state/display models.
- XNA collision code is simplistic.
  » Bounding boxes
  » Bounding spheres
  » Planes, lines, etc.
- In simulations, center of mass and point of incidence affect rotation of objects.
Collisions

• Doing proper collisions is extremely tough.
  – In simulations, center of mass and point of incidence affect rotation of objects.
  – In games, ‘close enough’ is usually sufficient.
    » Use primitive shapes that approximate the overall shape of the object.
    » Use existing algorithms – easy to miss boundary cases (pun intended).
Collisions / Reflections

- When ‘bouncing’ objects off of each other, you need to:
  - Detect the collision
  - Detect the angle of incidence.
Collisions / Reflections

- Angle of incidence for circles is related to the tangent and normal vectors.
  - Forces transmit between shapes along this vector – see conservation of momentum
Collisions / Reflections

- For fixed surfaces and elastic collisions, the object will bounce off the surface with the same angle as it hits:
For fixed surfaces and elastic collisions, the object will bounce off the surface with the same angle as it hits:
Collisions / Reflections

- To compute the reflection, you need the velocity vector and the surface normal vector (shown).
Collisions / Reflections

- \( R = V - 2(N)(V \cdot N) \)
  - (Use dot product)
  - \( N \) must be normalized (divide both components of \( N \) by the length of \( N \) to make a unit vector)
  - \( N \) must be perpendicular to the surface, but either \( N \) or \(-N\) will work.
  - Use unit vector for \( N \). (Normalize it.)

- The reflection vector is the reflected velocity.
Next assignment

- Create a simulation where a ball bounces around a field of fixed objects
- Simulation should resemble a pinball game
- Must use semi-random velocity for initial state of ball
- Must use XNA
- Details posted tomorrow.