Zenipex Library 3D

Mitchell Keith Bloch

University of Michigan
2260 Hayward Street
Ann Arbor, MI. 48109-2121
bazald@umich.edu

October 2, 2013

Outline

2D Recap

- Game Objects
  - 2D representation of position
  - 2D representation of size/extents
  - 2D motion and interaction
  - 2D rendering
- *Can* use abstract coordinates rather than screen coordinates

3D Explained

- Game Objects
  - 3D representation of position
  - 3D representation of size/extents
  - 3D motion and interaction
  - 3D rendering
- Not all of the above are necessary, but some subset is
- *Must* use abstract coordinates rather than screen coordinates

2D to 3D

- `Point2f` → `Point3f`
  - Adds a `z` component
  - Integrates nicely with `Vector3f`
- `Vector2f` → `Vector3f`
  - Adds a `k` / `z` component
  - More featureful than `Vector2f`
- Angle (float) → `Quaternion`
  - Representing rotations is more difficult
- Greater complexity
  - Controls
  - Model of the World
  - Visualization

Vector3f

- Cannot ignore 'k' ('z') component in 3D
- Inadequate representation of extents
  - Not everything should be a box
  - Think in terms of geometric primitives
  - `Zeni::Collision` objects can help with intersection and distance tests

Geometric Primitives

- `Point3f`
- `Line`, `Ray`, and `Line_Segment`
- `Plane`
- `Sphere` (`Point3f` with radius)
- `Capsule` (`Line_Segment` with radius)
- `Infinite_Cylinder` (`Line` with radius)
- `Parallelepiped`
- Primitives can be combined to form more complex shapes

Outline

Much the Same

- Gamestate transition logic
- Basic input handling
- Sound (unless doing positional audio)
- Timing
- Heads-Up Display (HUD) rendering
  - Just follow `set_3d(...)` with `set_2d(...)`

Outline

Motion

- Linear motion is easy
  - `Point3f point = Point3f() + Vector3f();`
- Rotational motion is more complicated
  - `Quaternion`s are often used
  - Can convert to and from Axis-Angle representation

Quaternions

- `Quaternion::Axis_Angle(axis, angle)`
  - Careful, a similar-looking constructor is very different
- `Quaternion`s represent abstract rotations
  - Like angles, they do not “point” anywhere, but represent a change in orientation
  - A default orientation must be known for a rotation to be meaningful
- `Vector3f vector = Quaternion() * Vector3f();`
- `Quaternion`s are composable
  - `Vector3f v1 = r1 * r0 * v0;`

Collisions

- Collision detection is “easy”
  - While non-trivial to write, the algorithms are easily reused in different circumstances
- Collision response is trickier
  - Difficult to everything “right”
  - Using a few assumptions can help
    - Special-case collision detection with ground
  - Think hard about your requirements before starting implementation, and test thoroughly

Outline

Camera

- In 2D, ‘upper left’ and ‘lower right’ was enough
- Now we need
  - Position (`Point3f`)
  - Forward (`Vector3f`)
  - Up (`Vector3f`)
  - Field of view in ’y’ (float, radians)
  - Near clip (float, distance in the forward direction)
    - As a rule of thumb, keep at least `10.0f` to avoid Z-Buffer errors
  - Far clip (float, distance in the forward direction)
    - Keep small if possible, but less important than to keep near clip large

Frustum

- `Camera` represents a matrix transformation
- Perspective transformation is called a frustum
- Actually looks quite bad if you look closely
- Keep the player away from corner cases
  - Literally, distortion is highest near the corners of the screen, closest to the `Camera`
  - Ensure that the near clip isn't too close, and that objects are even further away

`Projector3D`

- `Projector3D` can be used to transform between coordinate systems
  - Screen coordinates with depth, [`0.0f`, `1.0f`]
  - World coordinates within the `Camera`'s frustum
- Can make a `Ray` pointing into the scene by converting screen coordinates with depth `0.0f` and `1.0f` to world coordinates
- Can also determine maximum distance to near clipping plane this way

Outline

Model

- Many triangles
  - Geometry preferred to transparency – z-buffer issues
- Entire `Model` is stored in a `Vertex_Buffer` object
- At runtime, you can set
  - Scaling, rotation, and translation
  - Keyframe for animation
    - But not mesh morphs
- Before runtime, using 3ds Max, individual triangles can be given materials

Materials

- A `Material` has
  - Ambient, diffuse, and specular `Color` channels
  - Emissive `Color` (innate glow)
  - Power/Shininess (affects sharpness of specular highlights)
  - Possibly a `Texture`
    - Must add to the `Textures` database, even for `Models`
- Missing (for now)
  - Bump mapping, displacement mapping, ...

Lighting

- Lights have the same diffuse/ambient/specular channels as Materials
  - Color values multiply ($\text{Light}_{\text{Diffuse}} * \text{Material}_{\text{Diffuse}}$, ...)
- Lights can be point sources, directional, or spotlights
- Lights do not cast shadows
- Lights will slow down your program
- Spotlights look bad without insanely high numbers of triangles or shaders

Fog

- `Fog` is a distance-based haze
  - Not volumetric fog
- As distance from the `Camera` increases, objects become the color of the `Fog`
- It should probably match the `clear_color` of the screen, or come close to matching your background color

Warnings

- Collision response with concave objects difficult to do well
  - Crouching under stairs is tricky
- Rendering objects in fixed positions and orientations relative to the player is non-trivial
- Complex animation is difficult
  - 3ds animation is error prone
  - Some animation can be done in code
  - **Mesh morphs are unsupported**

Advice

- Good design and encapsulation is even more important in 3D game development than it was in 2D
- Think carefully about your coordinate systems
- Remember that you get to make many choices arbitrarily, but **which must be consistent!**
  - If a box is at a given size and orientation, its `Model` and its `Parallelepiped` must match
  - Rotations are from an arbitrary default orientation
  - Units of size have no default meaning