Lecture 15

Perspective in 2D Games
Graphics Lectures

- Drawing Images
  - SpriteBatch interface
  - Coordinates and Transforms

- Drawing Perspective
  - Camera
  - Projections

- Drawing Primitives
  - Color and Textures
  - Polygons

bare minimum to draw graphics
side-scroller vs. top down
necessary for lighting & shadows

2D Sprite Graphics
Take Away for Today

• What is the game “camera”?
  • How does it relate to screen space? Object space?
  • How does the camera work in a 2D game? 3D?

• How do we give 2D games depth?
  • Advantages, disadvantages of orthographic view
  • Advantages, disadvantages of axonometric view

• How does “tileability” affect art in games?
The Game Camera

- What makes a game 3-D?
  - Everything is shown on a 2-D screen (mostly)

- 3D game have a **user controlled** “camera”
  - Position camera to look at art from all sides
  - 3-D art has enough information to allow this

- CS/INFO 3152 limits you to a 2-D game
  - The game camera has a **fixed perspective**
  - You render all art to one visible side
Camera in 2D Games

Perspective
Specifying the Camera

- Camera is a **coord space**
  - Called “eye space”
  - Eye position at origin

- How to move camera?
  - Transforms again!

- **Inverse** of scrolling
  - **Scrolling**: move obj to eye
  - **Camera**: move eye to obj
  - Two matrices are *inverses*
Cameras in LibGDX

- LibGDX has a **Camera** class
  - Stores camera type, and eye location
  - We typically use **OrthographicCamera**
  - Define as size of screen, with origin at bottom

- Apply to SpriteBatch with **setProjection()**
  - Convert camera into a Matrix4 object
  - Use the **combined** field, **not** projection
  - See **GameCanvas.java** in **Lab 2**
Cameras in LibGDX

SpriteBatch batch = new SpriteBatch();

// Create a camera for the game window
Camera camera = new OrthogonalCamera(width,height);

// Set the camera in the SpriteBatch
Matrix4 matrix = camera.combined;
batch.setProjectionMatrix(matrix);

// Ready to use SpriteBatch
batch.begin();

...
## Cameras in LibGDX

<table>
<thead>
<tr>
<th>OrthogonalCamera</th>
<th>PerspectiveCamera</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Used for all 2D games</td>
<td>- Used for all 3D games</td>
</tr>
<tr>
<td>- Objects have 2d positions</td>
<td>- Objects have 3d positions</td>
</tr>
<tr>
<td>- Draws back-to-front</td>
<td>- Draws a picture plane</td>
</tr>
<tr>
<td><strong>Specify the viewport</strong></td>
<td><strong>Specify eye coordinates</strong></td>
</tr>
<tr>
<td>- The window size</td>
<td>- Eye origin</td>
</tr>
<tr>
<td>- The window origin</td>
<td>- Looking direction</td>
</tr>
<tr>
<td>- Move origin to scroll</td>
<td>- Up direction</td>
</tr>
</tbody>
</table>
Drawing for a Perspective

• 3D Models make it easy
  • Rotate model to position
  • Flatten to jpeg, tiff, etc…

• But 3D modeling is hard
  • Very technical programs
  • Cannot draw “by hand”

• How to draw perspective?
  • Artist “captures” camera

• **Realism creates problems**
The concept of the picture plane may be better understood by looking through a window or other transparent plane from a fixed viewpoint. Your lines of sight, the multitude of straight lines leading from your eye to the subject, will all intersect this plane. Therefore, if you were to reach out with a grease pencil and draw the image of the subject on this plane you would be “tracing out” the infinite number of points of intersection of sight rays and plane. The result would be that you would have “transferred” a real three-dimensional object to a two-dimensional plane.
Problem: Art assets are not invariant under translation.
Vanishing Points are **Not** Our Friend
Vanishing Points are Not Our Friend

Horizon Line
Eye Level
Vanishing Point
Vanishing Points are **Not** Our Friend
Vanishing Points are **Not** Our Friend

- Horizon Line
- Eye Level
- Vanishing Point

Should not be same
Vanishing Points are **Not** Our Friend

2D games rely on **distortional** perspectives
Orthographic Projection

- Project perpendicular to an axis
  - Top-down: perpendicular to z-axis
  - Side scrolling: perpendicular to y-axis

- Very easy to do artistically
  - Art objects are flat tiles
  - Layer tiles via compositing

- But enforces 2-D gameplay
  - 3rd dimension lost; cannot be seen
  - Distorted: All rays to eye are parallel
Orthographic Projection
Side-View: *Braid*
Top-Down: *Hotline Miami*
Top-Down: Gauntlet
Drawbacks of Orthographic Projection

- **Top-down** is extremely limiting
  - Can only see the top of the avatar
  - Hard to make interesting characters
  - Typically limited to platformers

- There little **no depth** to gameplay
  - At best can create gameplay *layers*
  - 3rd dimension is very discrete (2.5D)
  - Represent 3rd dimension with *parallax*
Parallax Scrolling

• Gives depth to orthographic projection
  • Objects in background have distance
  • Rate of scrolling depends on distance

• Implement with multiple background layers
  • Each layer scrolls at a different rate
  • See course website for sample code

• Often requires some degree of transparency
  • Composite front layers with back layers
Parallax Scrolling
Parallax Scrolling
Axonometric Projection

- Off axis view of object
  - View along all 3-axes

- Once again: **distorted**
  - Not a true projection
  - No *vanishing point*
  - Axes are “foreshortened”

- Allows 3-D gameplay
  - “Cliffs” are visible
  - May also *hide objects*!
Axonometric Projection
Axonometric Projection
Projection Types

- **Isometric**
  - All axes are equal
  - If need all dimensions
  - Used in classic RPGs

- **Dimetric**
  - $z$-axis is very short
  - $x, y$ axes are equal
  - Orthographic + depth
  - For aesthetic reasons only
Projection Geometry

• Axes relative to screen
  • \( z \) goes “into” the screen
  • \( x, y \) are in screen plane

• Axonometric coordinates
  • May not be “true” coords
  • “Meaning” of \( x, y, z \)?

• Orthographic substitutes
  • **Side-scroller**: \( y \) is height
  • **Top-down**: \( z \) is height

- **Perspective**
  - \( z \) is “artificial” dimension
Isometric View

- $x, y, z =$ Axonometric Coords
- $x', y' =$ Screen Coordinates

\[
x' = x - z \\
y' = y + \frac{1}{2}(x+z)
\]

Game View
Top View
Side View
Isometric View: Zaxxon
Dimetric View (Side-Depth)

- $x, y, z = \text{Axonometric Coords}$
- $x', y' = \text{Screen Coordinates}$

$$x' = x + \frac{1}{2}(z)$$
$$y' = y + \frac{1}{4}(z)$$
Dimetric View (Top-Depth)

- $x, y, z = \text{Axonometric Coords}$
- $x', y' = \text{Screen Coordinates}$

$x' = x + \frac{1}{4}(z)$
$y' = y + \frac{1}{2}(z)$

Game View
Top View
Side View
But **Gameplay** is Still Orthographic
But **Gameplay** is Still Orthographic

But formula allow us to “fly” between layers
Dimetric: *Black Friday*
Dimetric: Black Friday

Shadow is “position”
Dimetric: *Enter the Gungeon*
Dimetric: *Enter the Gungeon*
Drawing for Axonometric View

- Use boxes shown on slide
  - Tiling boxes is easy
  - Draw shape inside box

- Complex, large shapes?
  - Glue together boxes
  - Draw inside box group

- Objects need many angles
  - Transparency is tricky
  - Standard: 8 compass points

- **Example**: LakeHills.ai
Drawing for Axonometric View

- Use boxes shown on slide
  - Tiling boxes is easy
  - Draw shape inside box

- Complex, large shapes?
  - Glue together boxes
  - Draw inside box group

- Objects need many angles
  - Transparency is tricky
  - Standard: 8 compass points

- **Example**: LakeHills.ai
Isometric Walking Animation
Isometric Walking Animation
## Which Style to Use?

<table>
<thead>
<tr>
<th>Orthographic</th>
<th>Axonometric</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Advantages</strong></td>
<td><strong>Advantages</strong></td>
</tr>
<tr>
<td>- Easy to make tiles</td>
<td>- Sort of easy to tile</td>
</tr>
<tr>
<td>- Easy to composite</td>
<td>- Some 3-D movement</td>
</tr>
<tr>
<td><strong>Disadvantages</strong></td>
<td><strong>Disadvantages</strong></td>
</tr>
<tr>
<td>- Movement is 2D</td>
<td>- Harder to composite</td>
</tr>
<tr>
<td>- Game feels flat</td>
<td>- Objects may be hidden</td>
</tr>
<tr>
<td>- Common in this class</td>
<td>- Lot of work for artist</td>
</tr>
</tbody>
</table>
Combining the Perspectives
Combining the Perspectives

Dimetric Environment

Orthographic Characters
Summary

- Camera represents “eye space” coordinates
  - 3D games have arbitrary camera movement
  - 2D games are limited to scrolling movement

- 2-D art requires you chose a projection
  - **Orthographic** is easy, but limits gameplay
  - **Axonometric** has better gameplay, but harder to draw

- Axonometric type depends on style of game
  - Isometric common to classic RPGs
  - Dimetric gives depth to traditional orthographic