

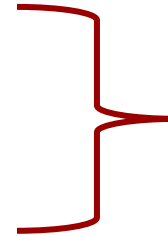
Lecture 14

Perspective in 2D Games

Graphics Lectures

- Drawing Images

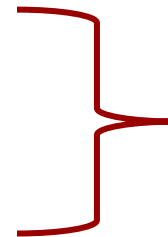
- SpriteBatch interface
- Coordinates and Transforms



bare minimum
to draw graphics

- **Drawing Perspective**

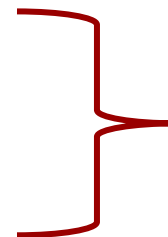
- Camera
- Projections



side-scroller vs.
top down

- Drawing Primitives

- Color and Textures
- Polygons



necessary for
lighting & shadows

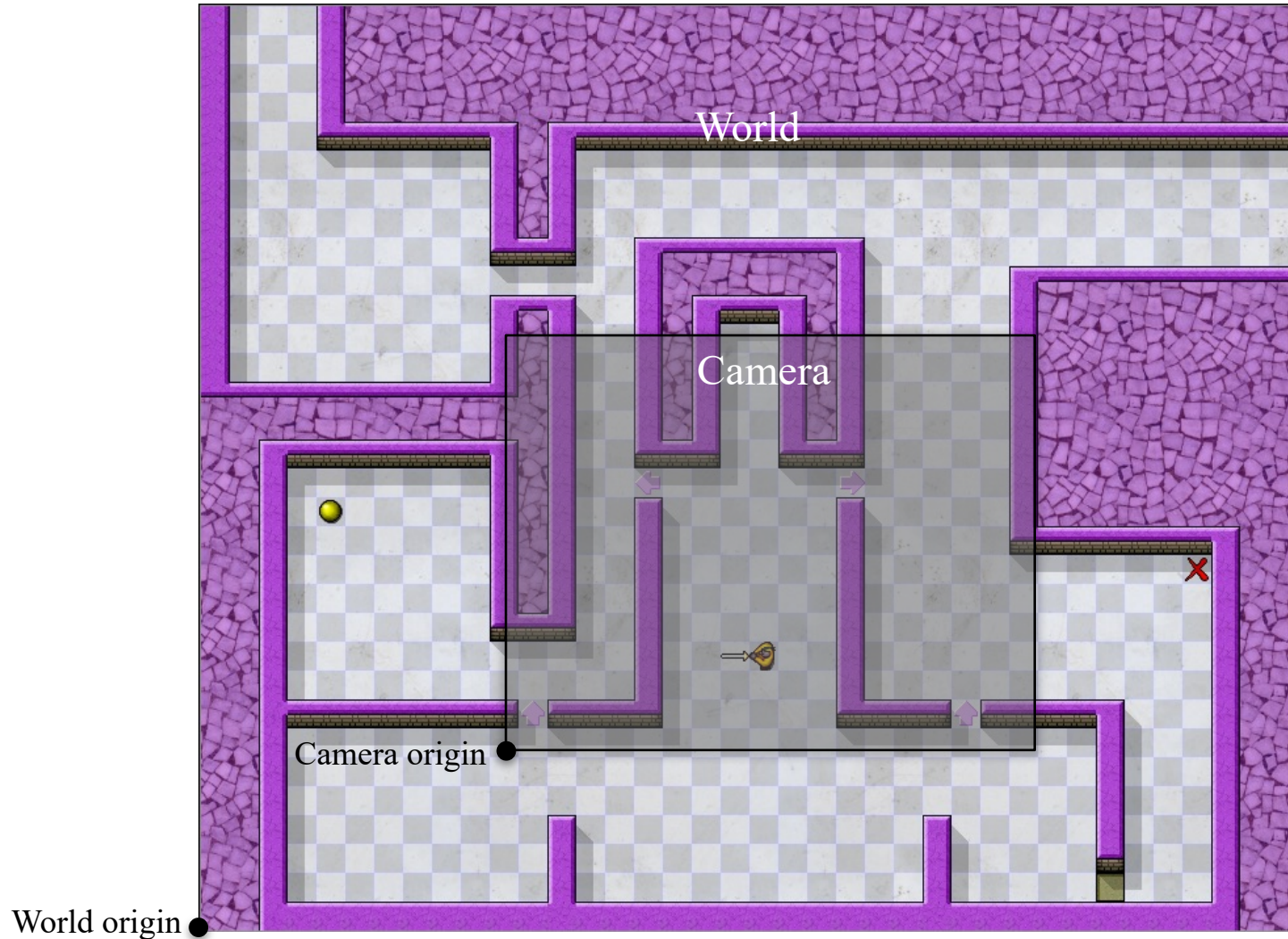
Take Away for Today

- 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

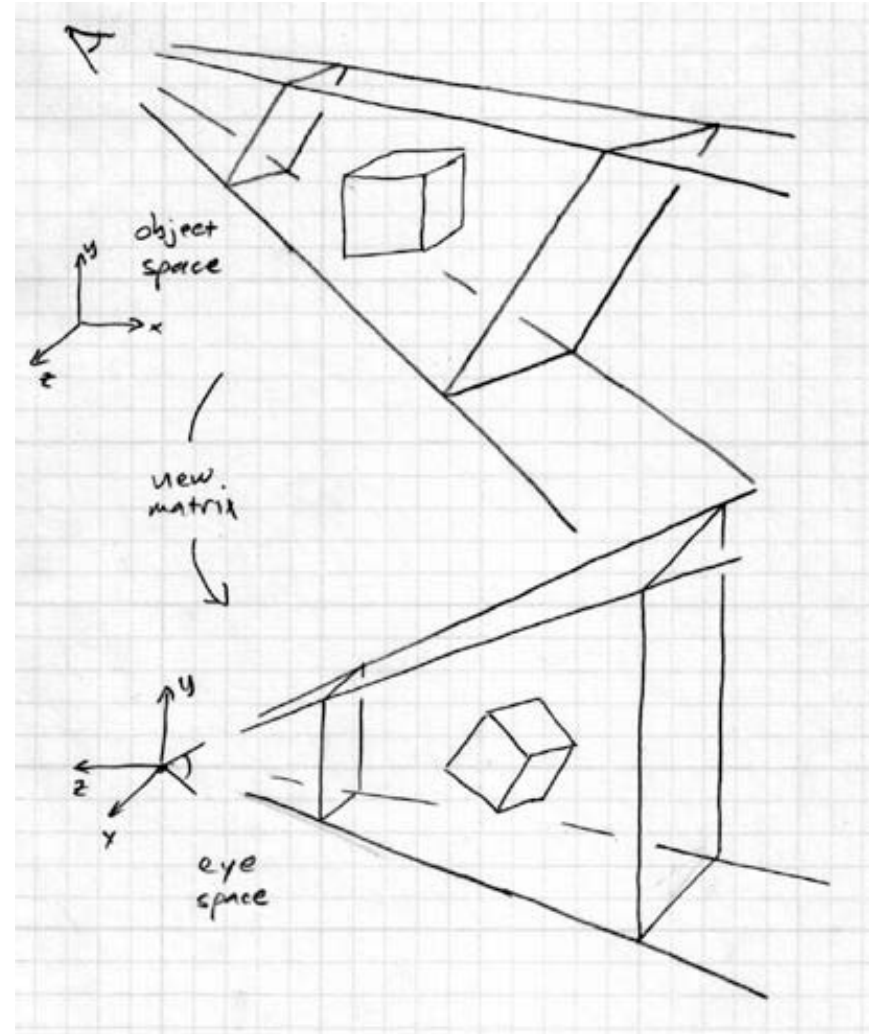
- What makes a game 3-D?
 - Everything is shown on a 2-D screen (mostly)
- 3D game have a **user controlled** “camera”
 - Position camera to look at art from all sides
 - 3-D art has enough information to allow this
- CS/INFO 3152 limits you to a 2-D game
 - The game camera has a *fixed perspective*
 - You render all art to one visible side

Camera in 2D Games



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 OrthographicCamera(width,height);
```

```
// Set the camera in the SpriteBatch
```

```
Matrix4 matrix = camera.combined;
```

```
batch.setProjectionMatrix(matrix);
```

```
// Ready to use SpriteBatch
```

```
batch.begin();
```

```
...
```



Convert Camera to
transform to use

Cameras in LibGDX

OrthographicCamera

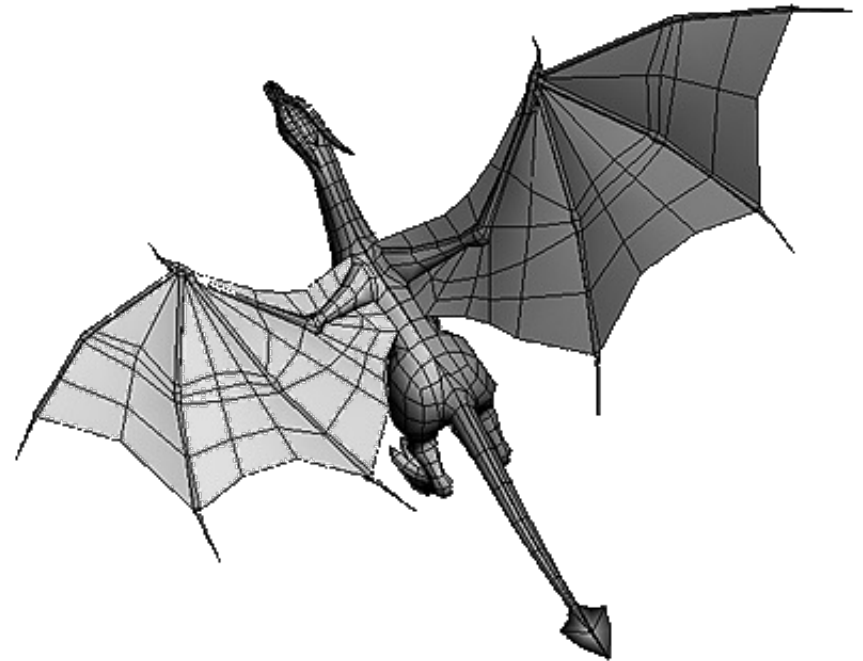
- Used for all 2D games
 - Objects have 2d positions
 - Draws back-to-front
- Specify the *viewport*
 - The window size
 - The window origin
 - Move origin to scroll

PerspectiveCamera

- Used for all 3D games
 - Objects have 3d positions
 - Draws a picture plane
- Specify *eye coordinates*
 - Eye origin
 - Looking direction
 - Up direction

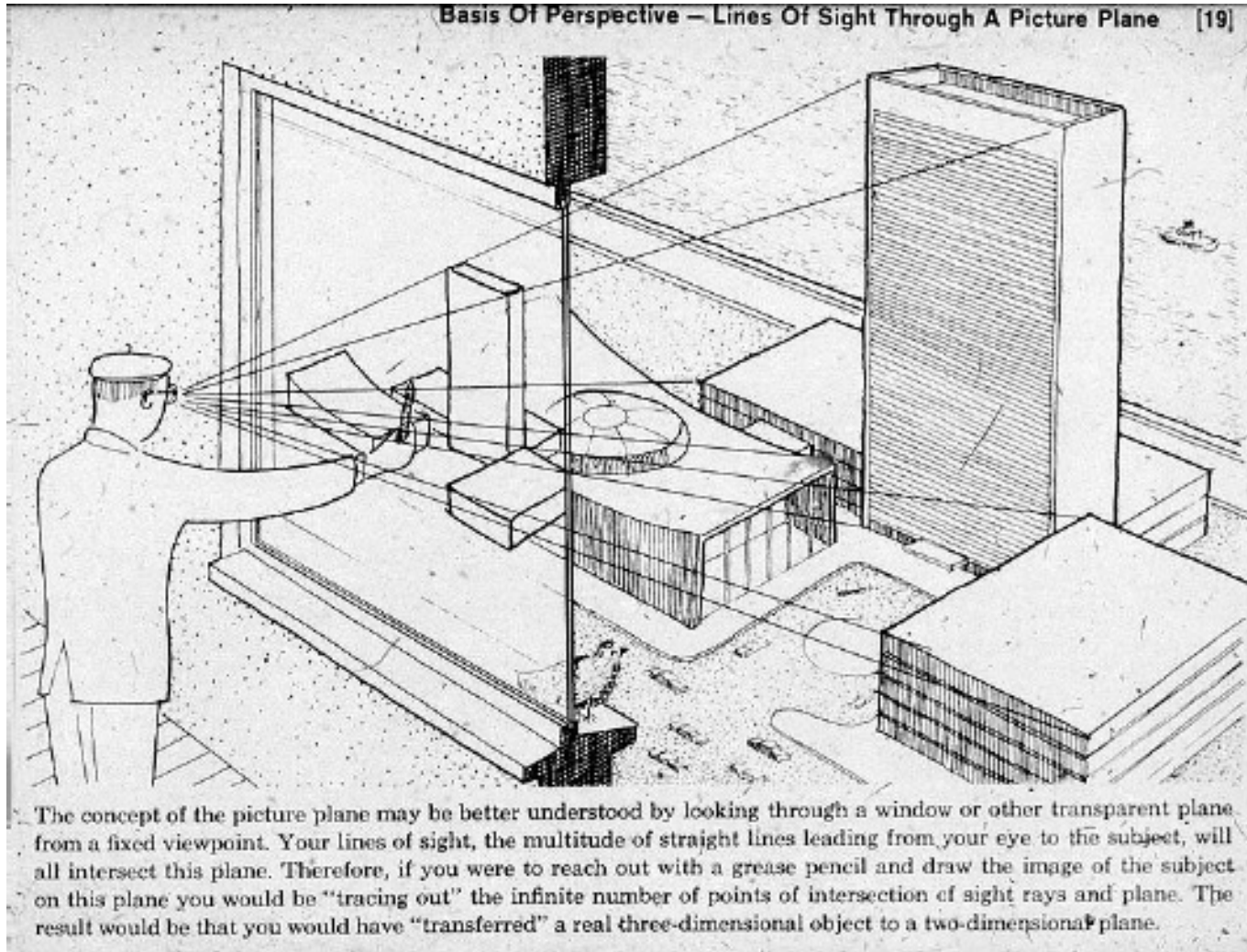
Drawing for a Perspective

- 3D Models make it easy
 - Rotate model to position
 - Flatten to png, tiff, etc...
- But 3D modeling is hard
 - Very technical programs
 - Cannot draw “by hand”
- How to draw perspective?
 - Artist “captures” camera
- **Realism creates problems**



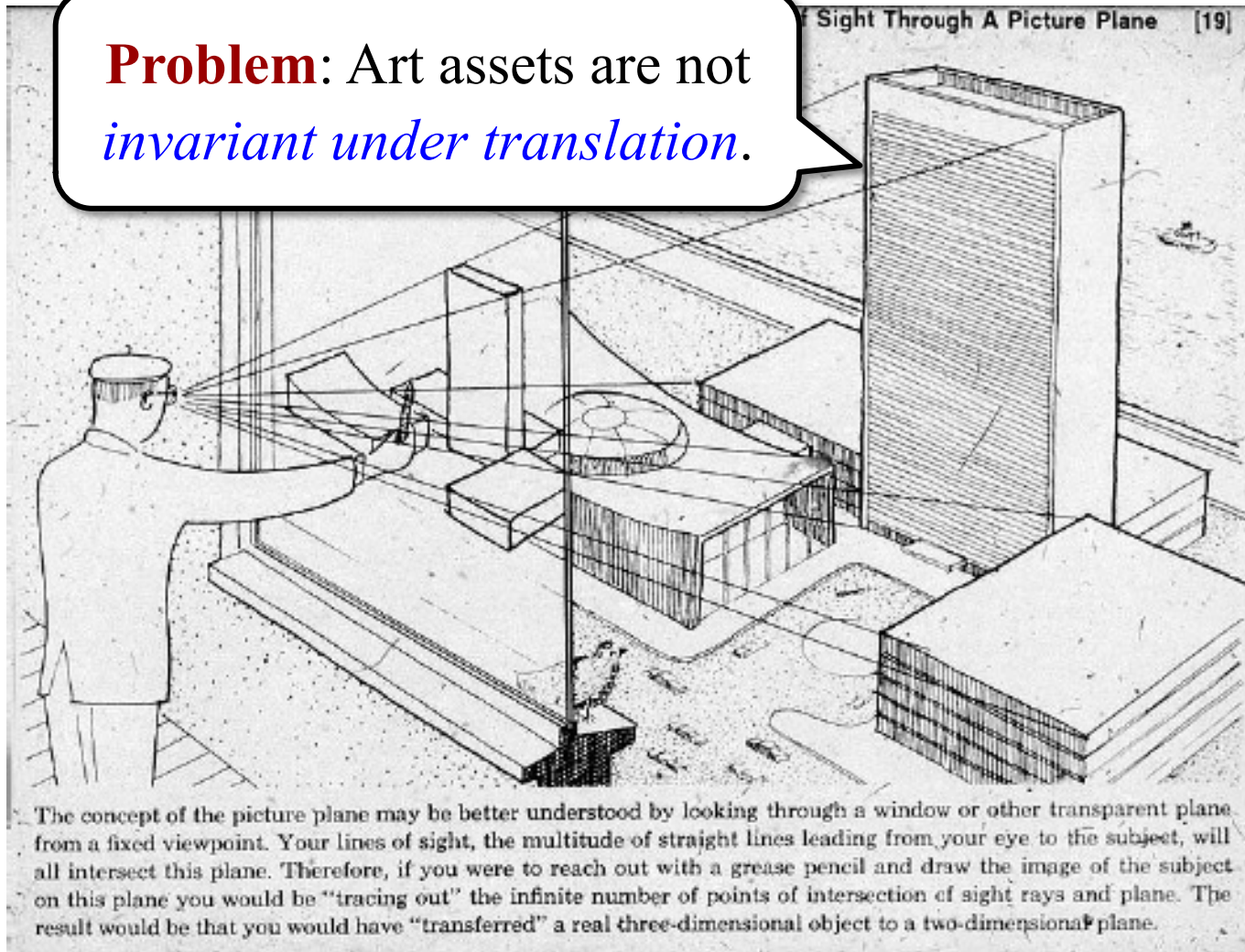
© 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025

Plane Projection in Drawing

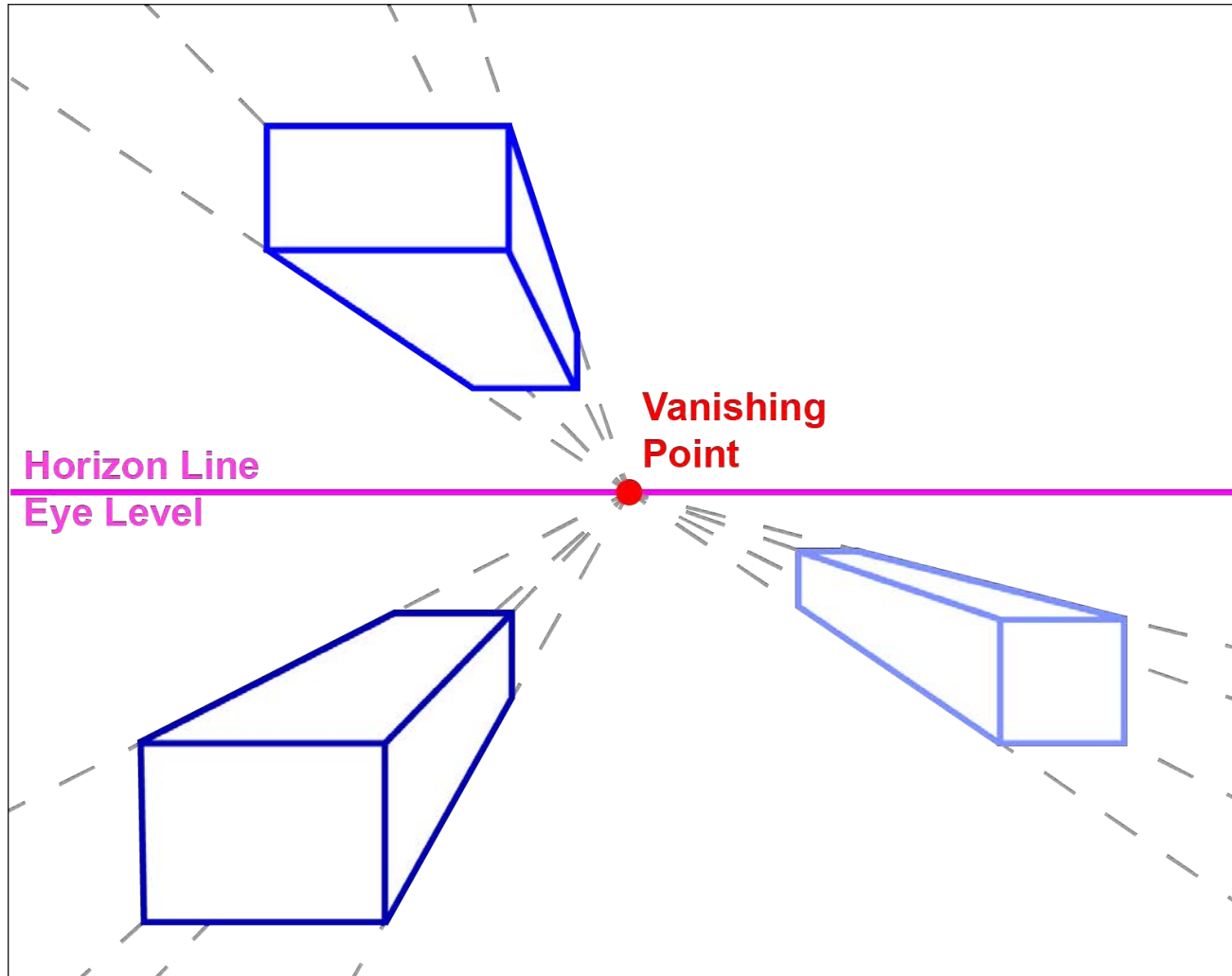


Plane Projection in Drawing

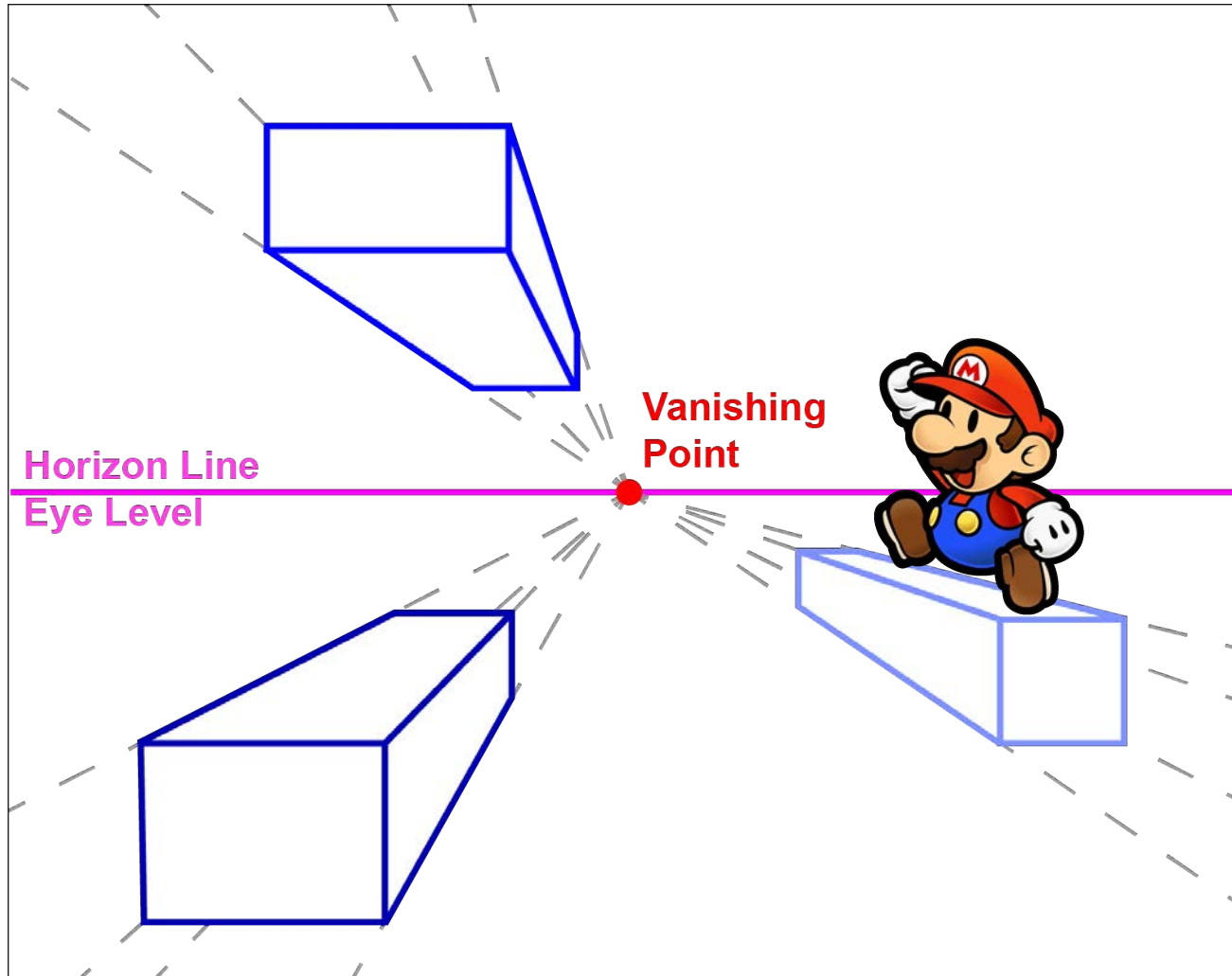
Problem: Art assets are not *invariant under translation*.



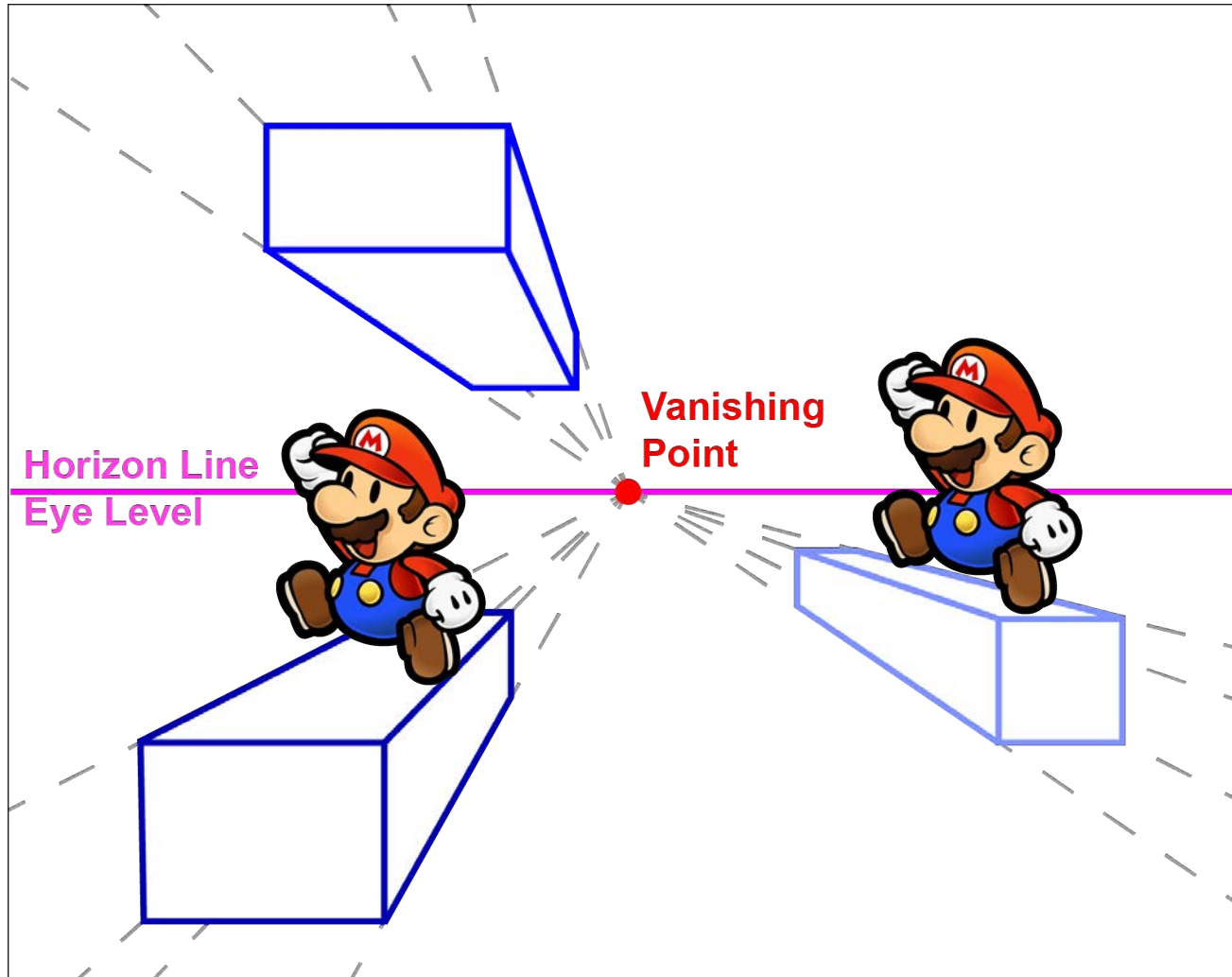
Vanishing Points are **Not** Our Friend



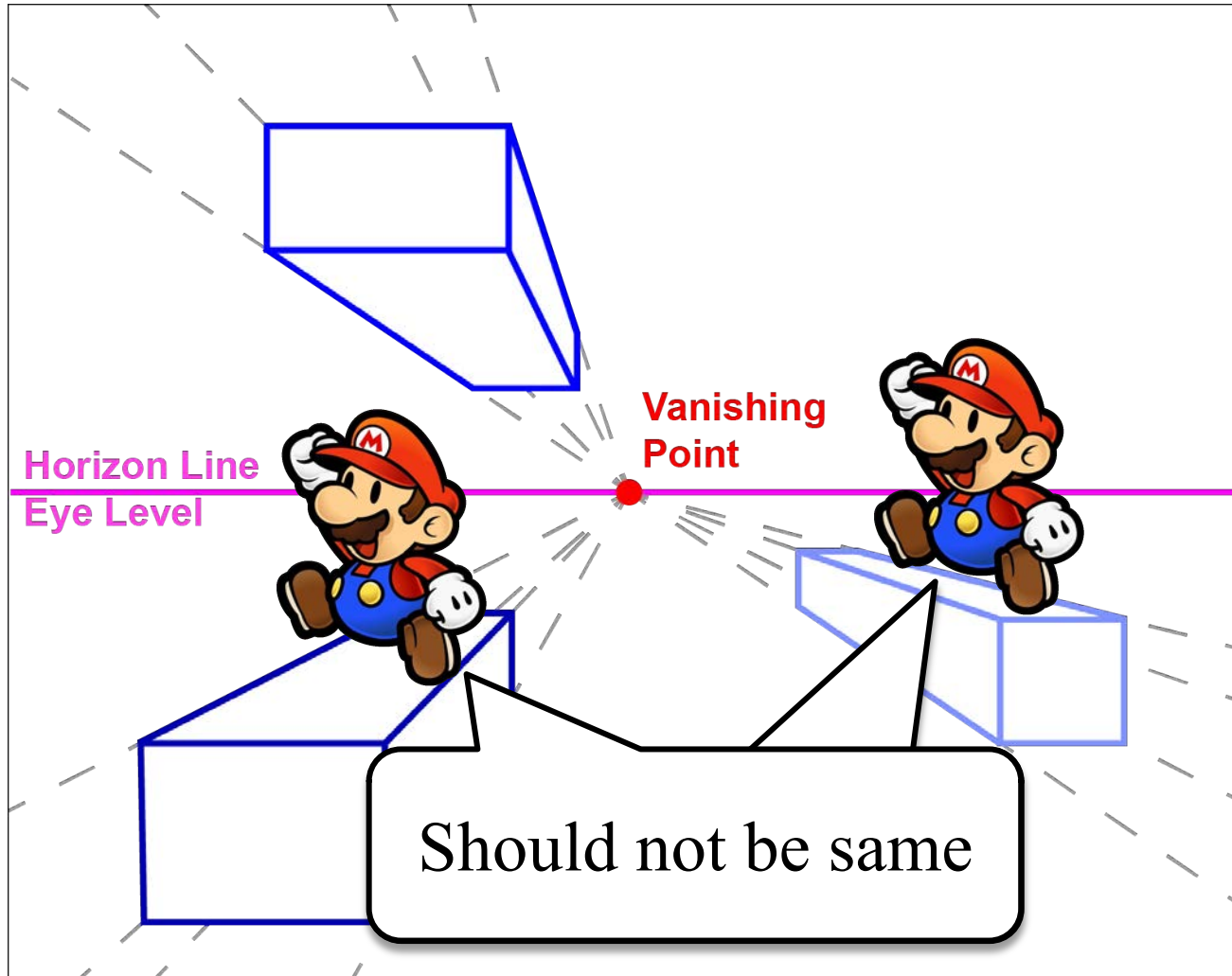
Vanishing Points are **Not** Our Friend



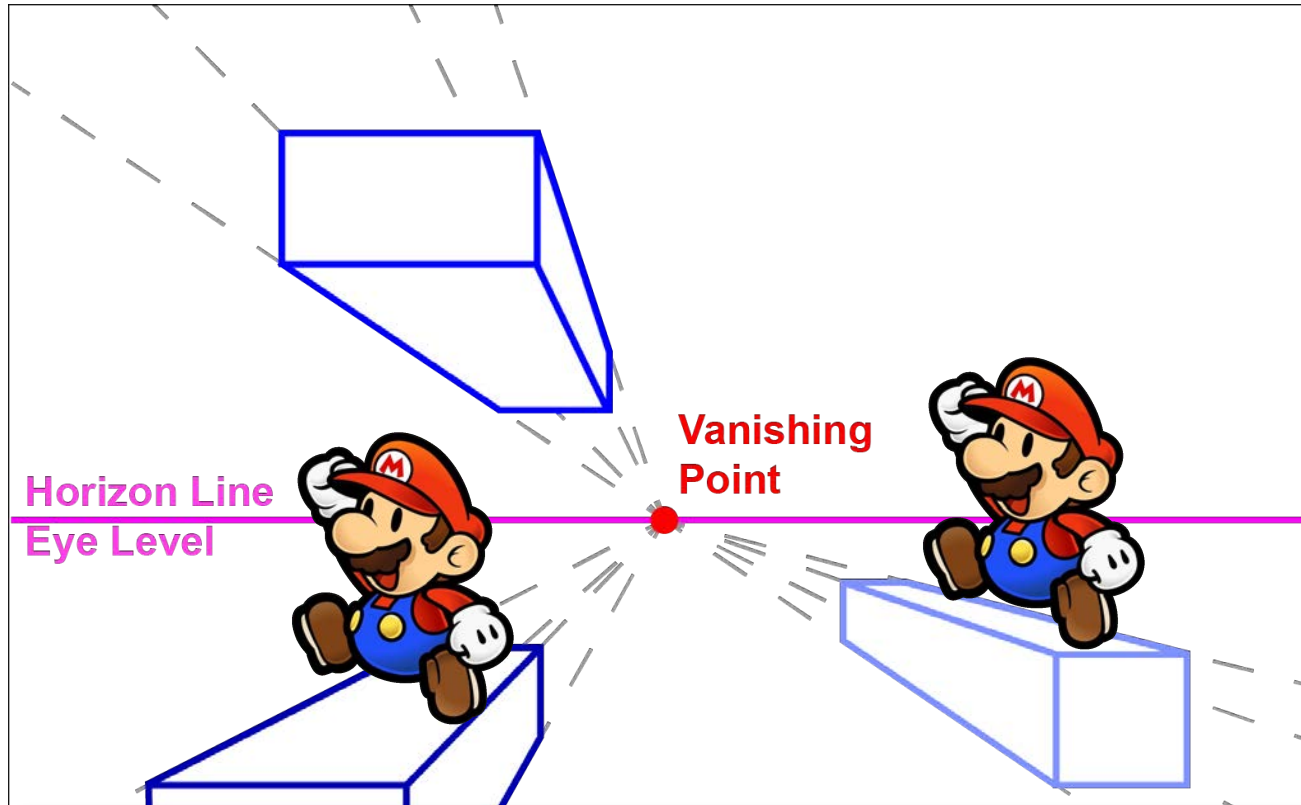
Vanishing Points are **Not** Our Friend



Vanishing Points are **Not** Our Friend



Vanishing Points are **Not** Our Friend



2D games rely on **distortional** perspectives

Parallel Projections

- Sprite art requires parallel projections
 - Parallel lines are always parallel on screen
 - Images can be translated within projection
- Three basic types of parallel projections
 - **Orthographic** (reading calls this *Multiview*)
 - **Axonometric**
 - **Oblique** (particularly *Cabinet*)
- See today's reading for taxonomy

Parallel Projections

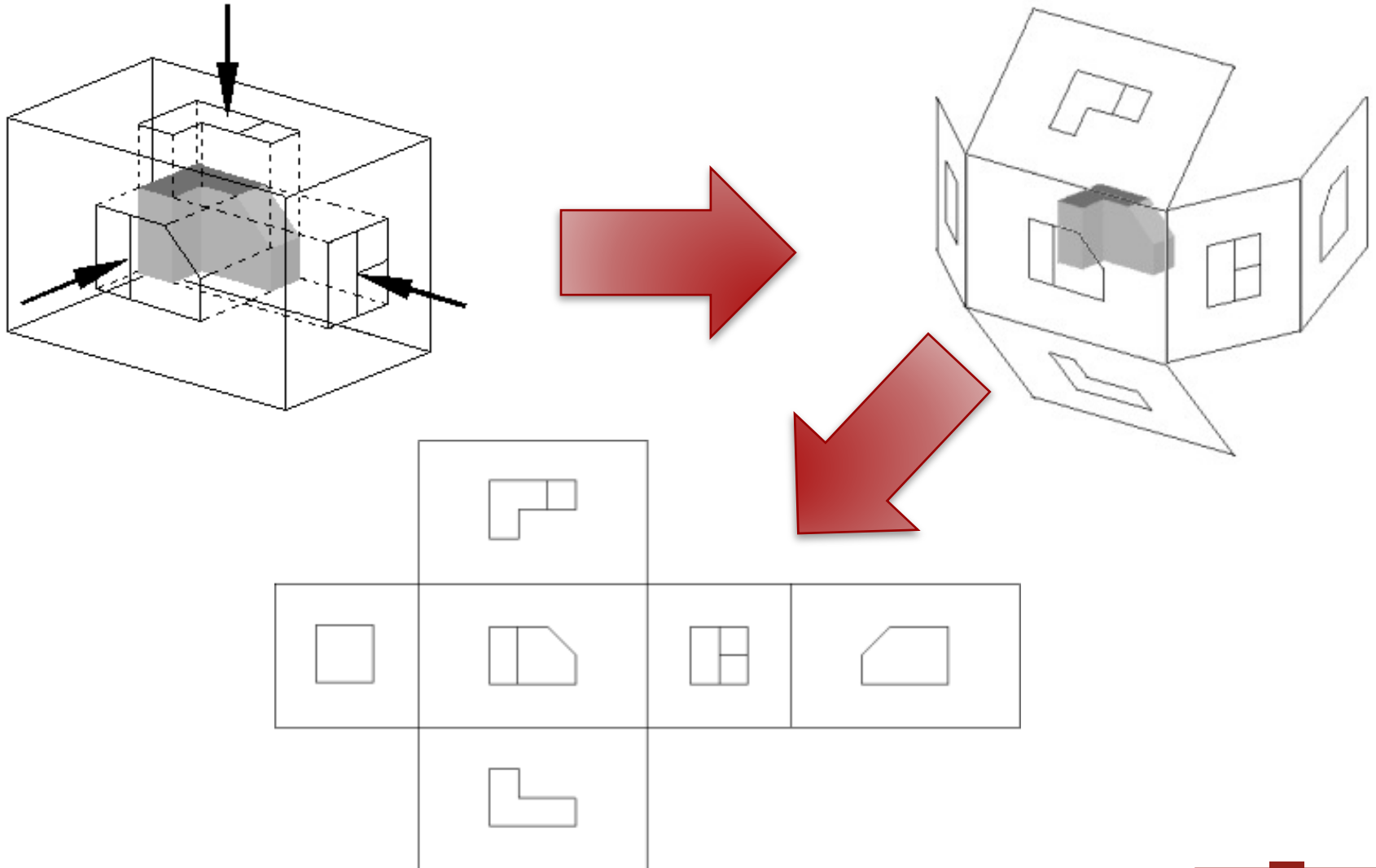
- Sprite art requires parallel projections
 - Parallel lines are always parallel on screen
 - Images can be translated within projection
- Three basic types of parallel projections
 - **Orthographic** (real world)
 - **Axonometric**
 - **Oblique** (particular)
- See today's reading for taxonomy

Not everyone uses these terms in the exact same way.

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



Perspective

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*

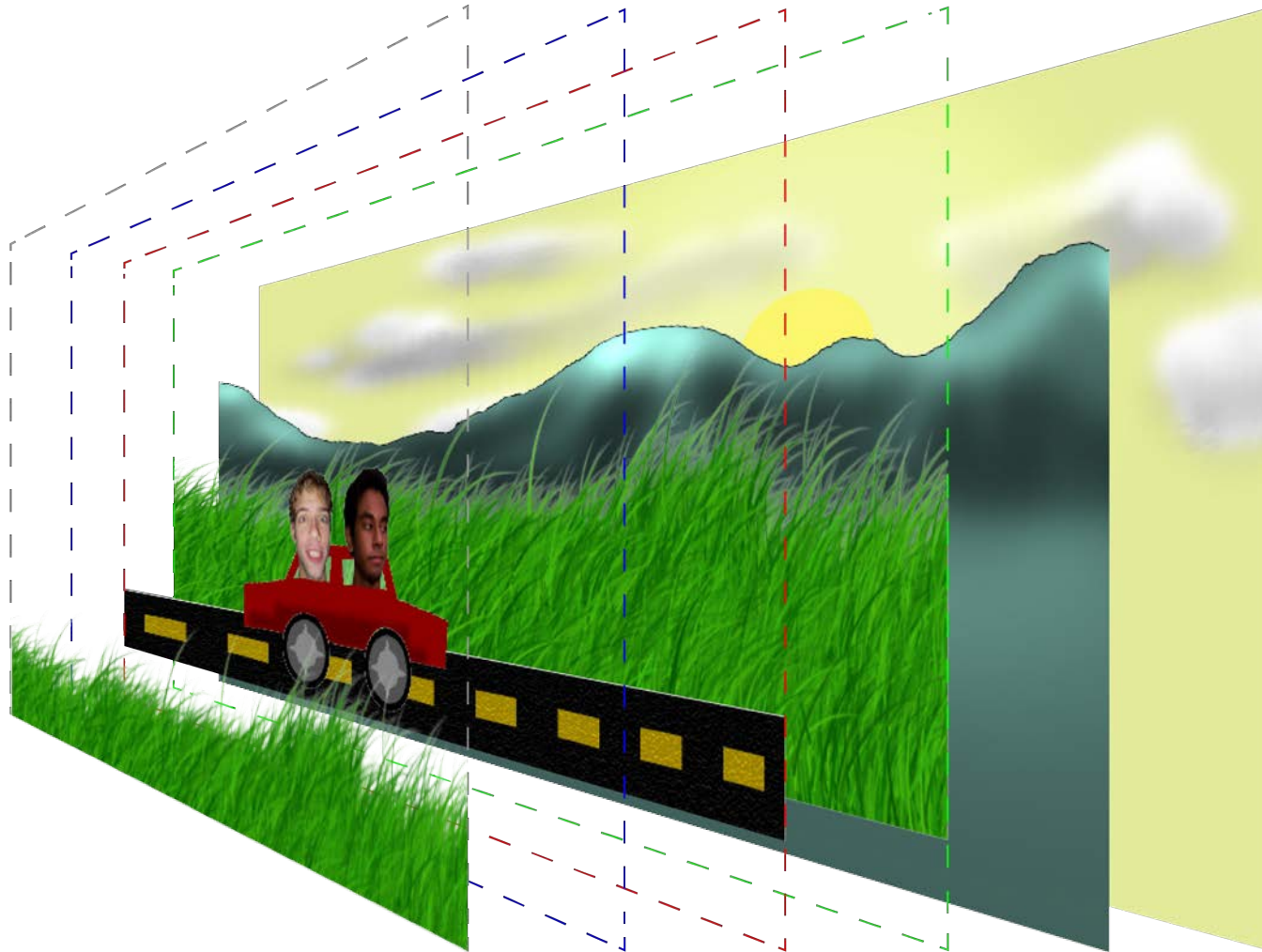
Parallax Scrolling

- 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

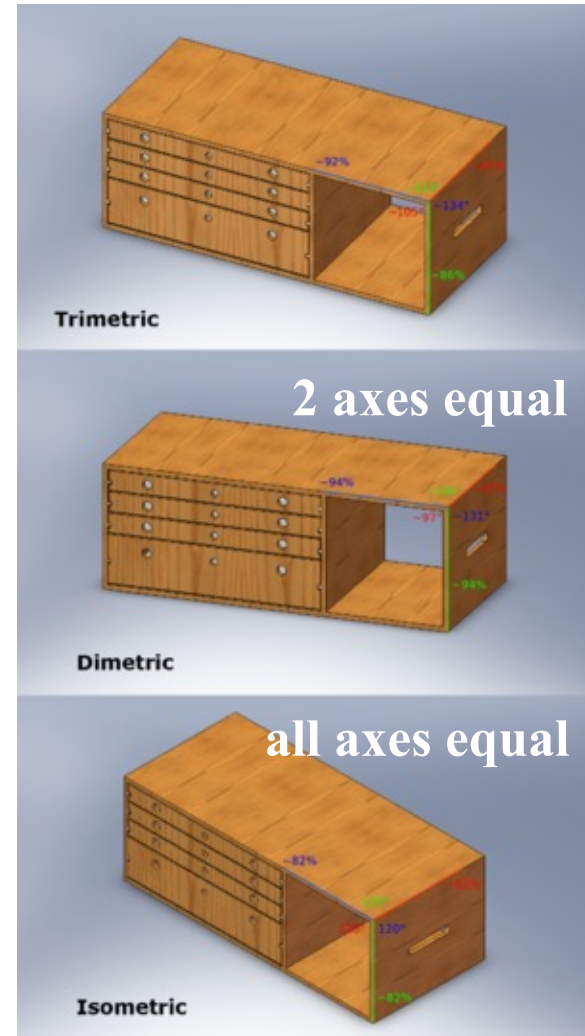


Parallax Scrolling



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: *Starcraft*

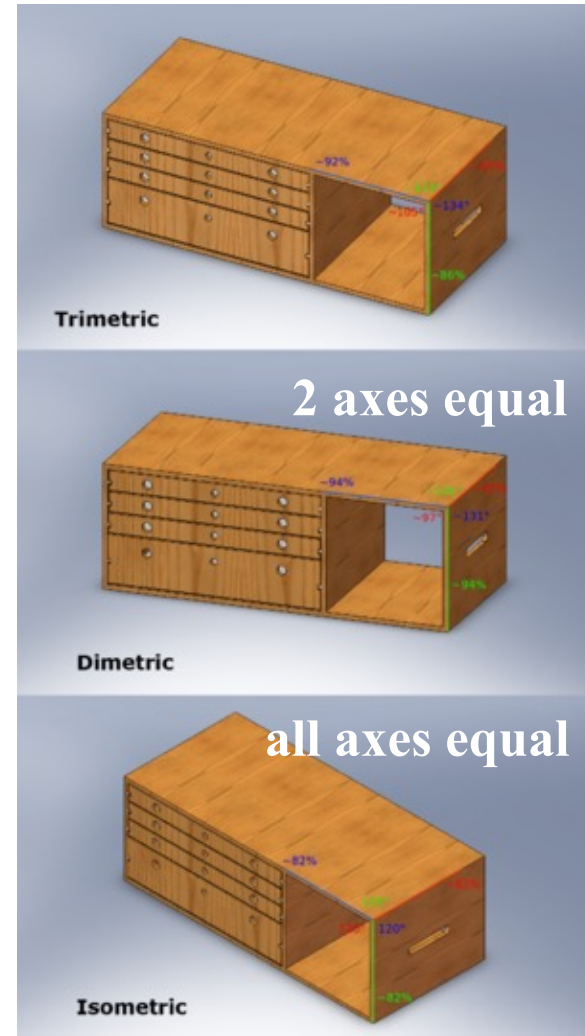


Isometric: *Avernum* Series



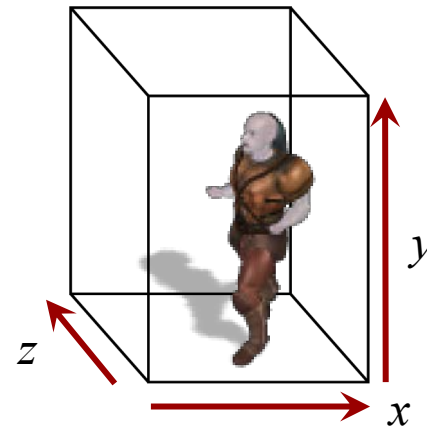
Projection Types

- **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



Isometric

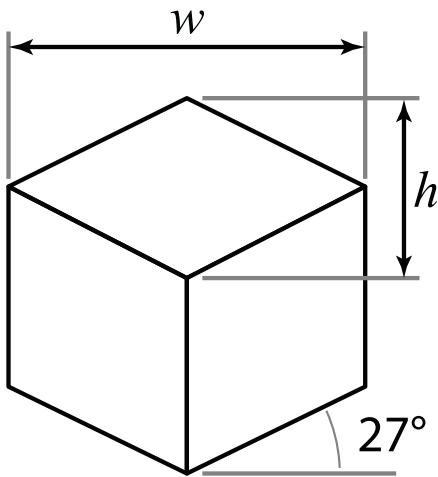
z is “artificial”
dimension

Isometric View

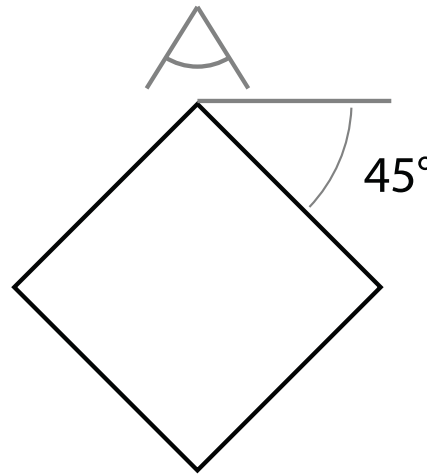
- $x, y, z =$ Axonometric Coords
- $x', y' =$ Screen Coordinates

$$x' = x - z$$

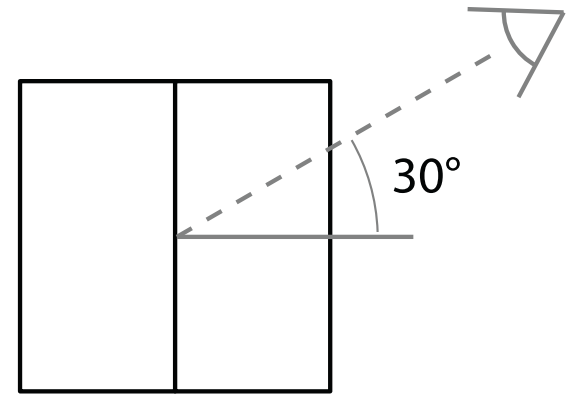
$$y' = y + \frac{1}{2}(x+z)$$



Game View

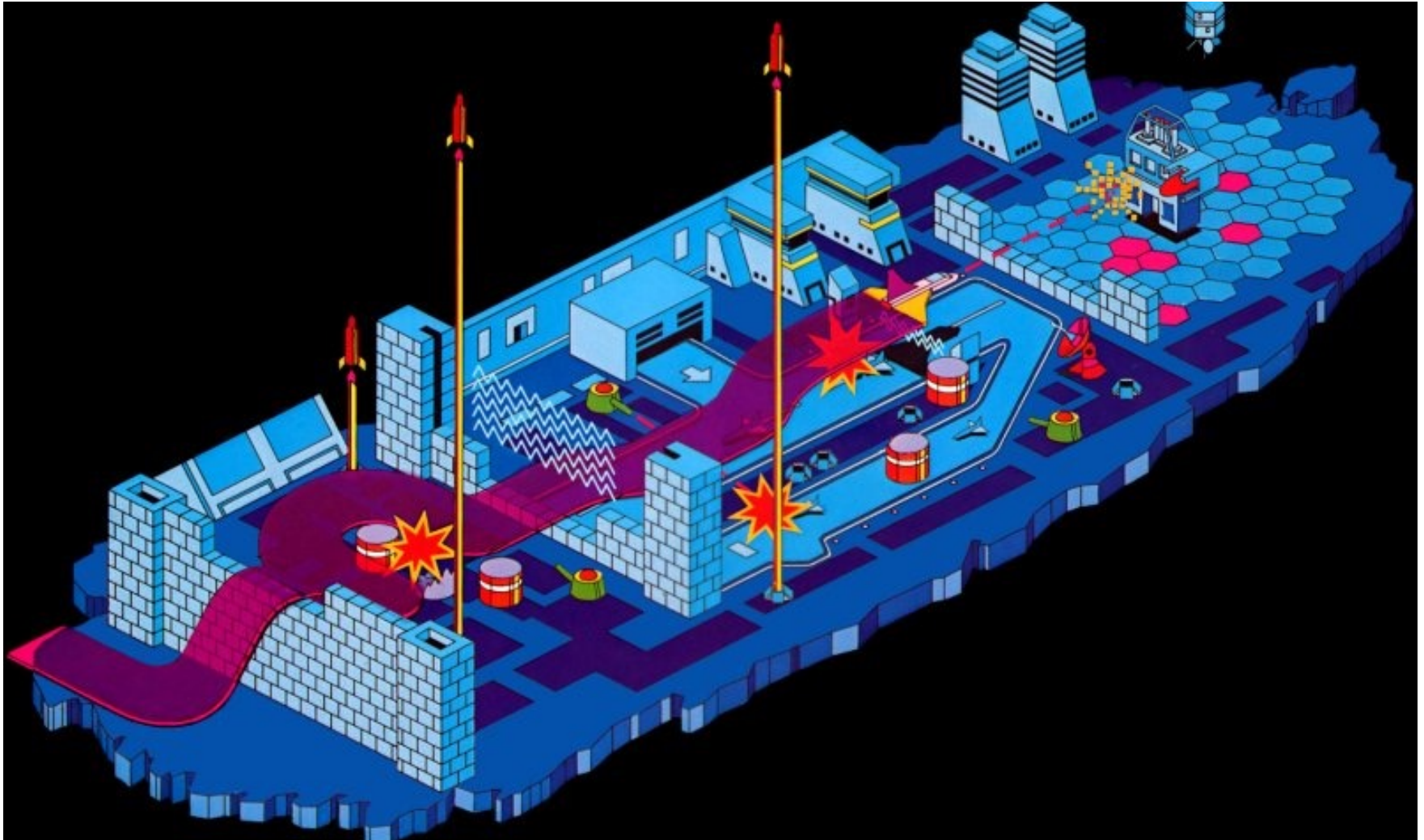


Top View



Side View

Isometric View: Zaxxon

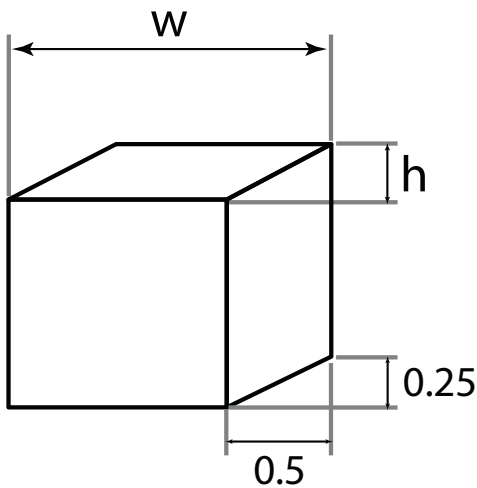


Classic Dimetric View (Side-Depth)

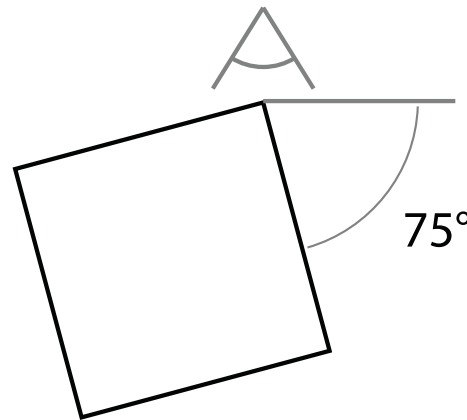
- $x, y, z =$ Axonometric Coords
- $x', y' =$ Screen Coordinates

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

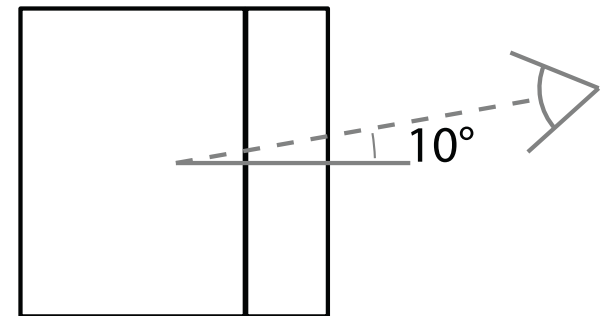
$$y' = y + \frac{1}{4}(z)$$



Game View



Top View

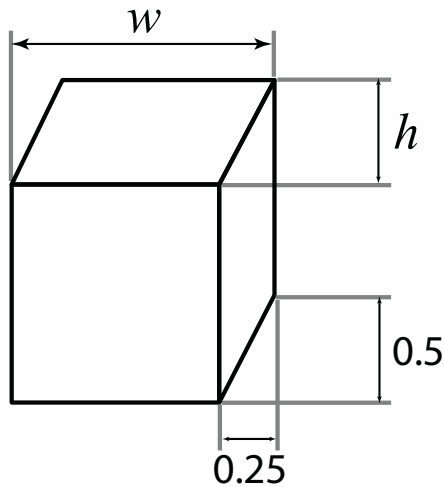


Side View

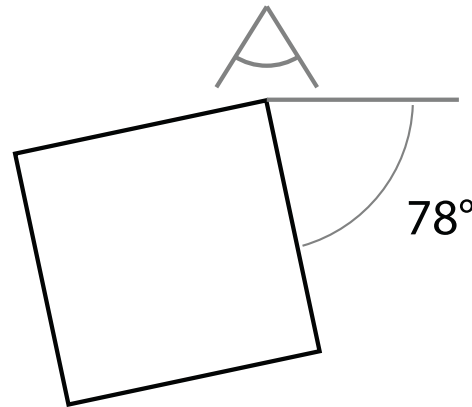
Classic Dimetric View (Top-Depth)

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

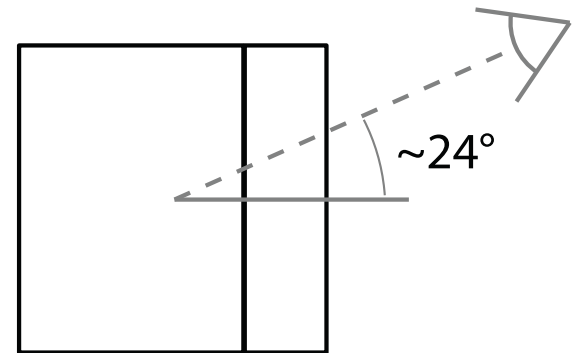
$$x' = x + \frac{1}{4}(z)$$
$$y' = y + \frac{1}{2}(z)$$



Game View



Top View



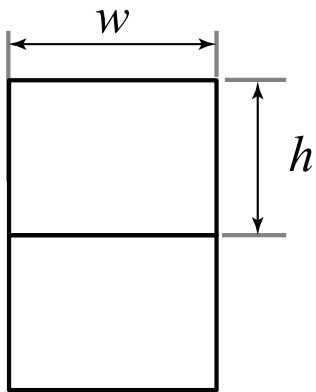
Side View

RPG “ $\frac{3}{4}$ Perspective”

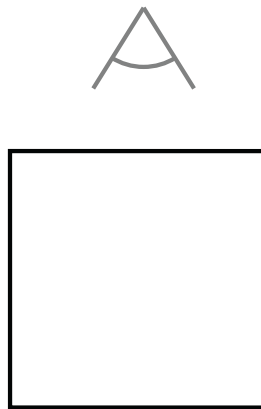
- $x, y, z =$ Axonometric Coords
- $x', y' =$ Screen Coordinates

$$x' = x$$

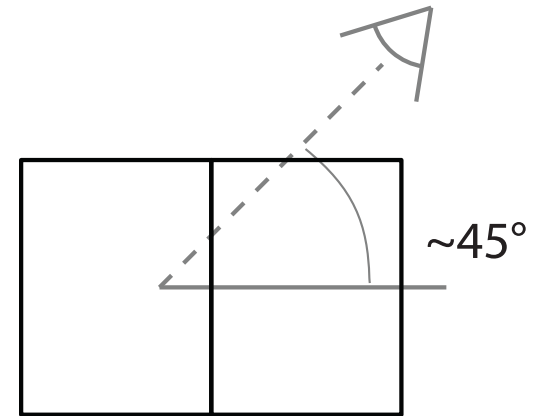
$$y' = \frac{3}{4}(y) + \frac{3}{4}(z)$$



Game View



Top View



Side View

$\frac{3}{4}$ Perspective: *Link to the Past*



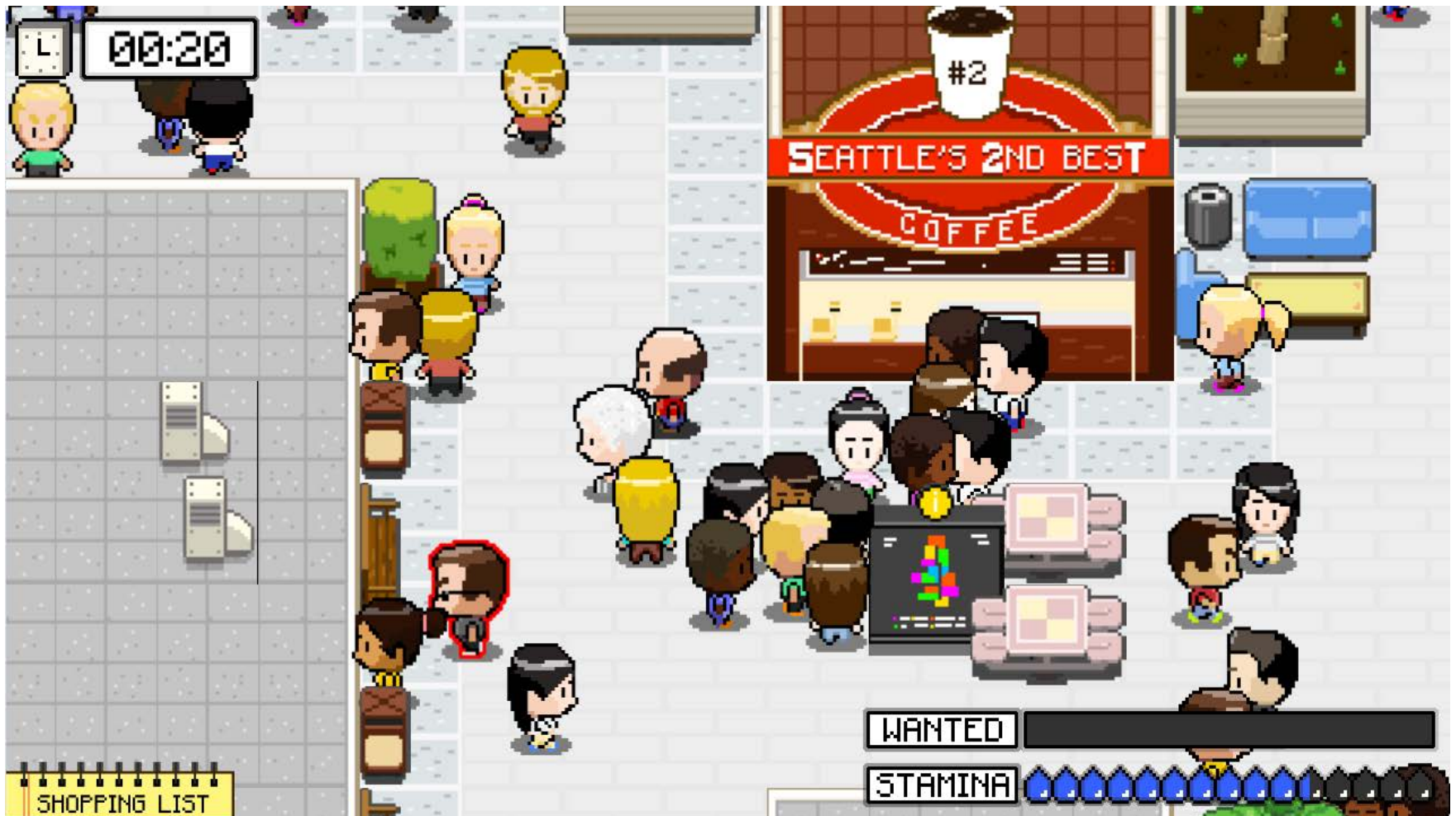
But **Gameplay** is Still Orthographic



But Gameplay is Still Orthographic



Dimetric: *Black Friday*



Dimetric: *Black Friday*



Dimetric: *Enter the Gungeon*



Dimetric: *Enter the Gungeon*

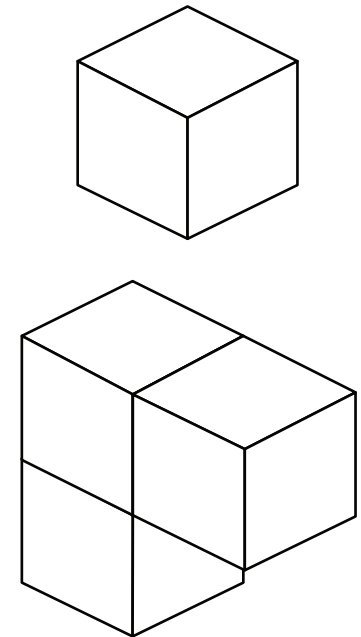


Isometric: *Baldur's Gate II*



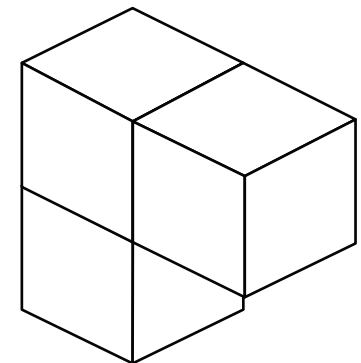
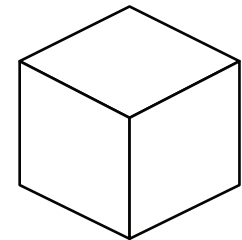
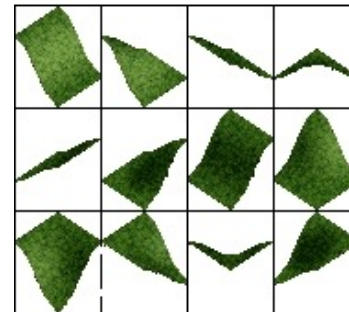
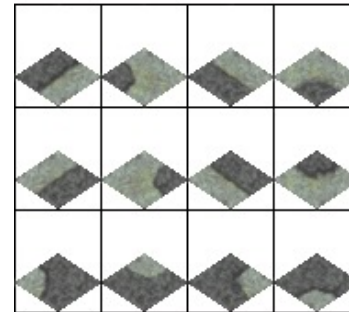
Drawing for Axonometric View

- 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

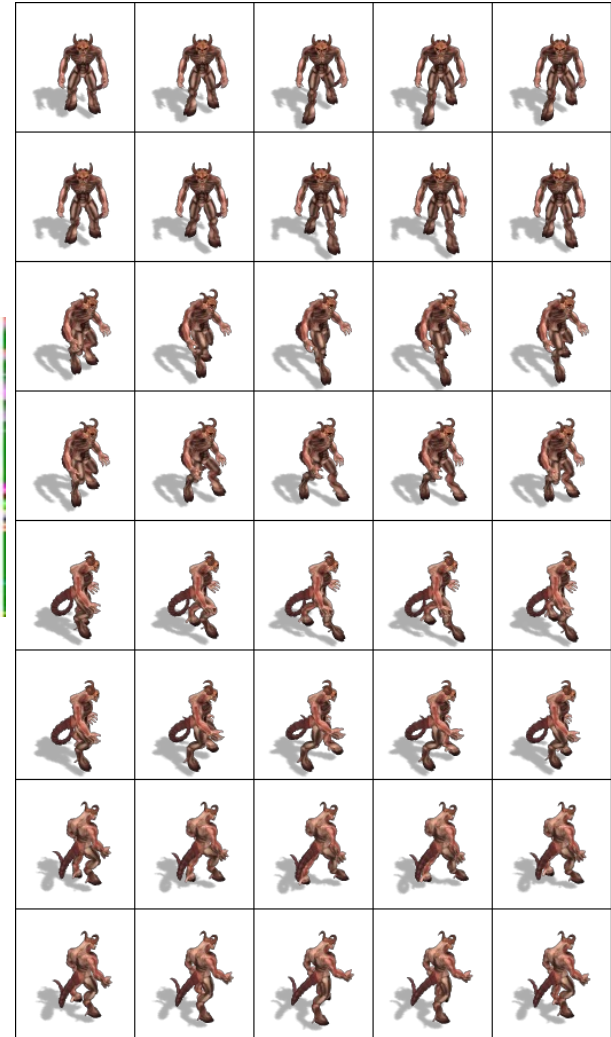
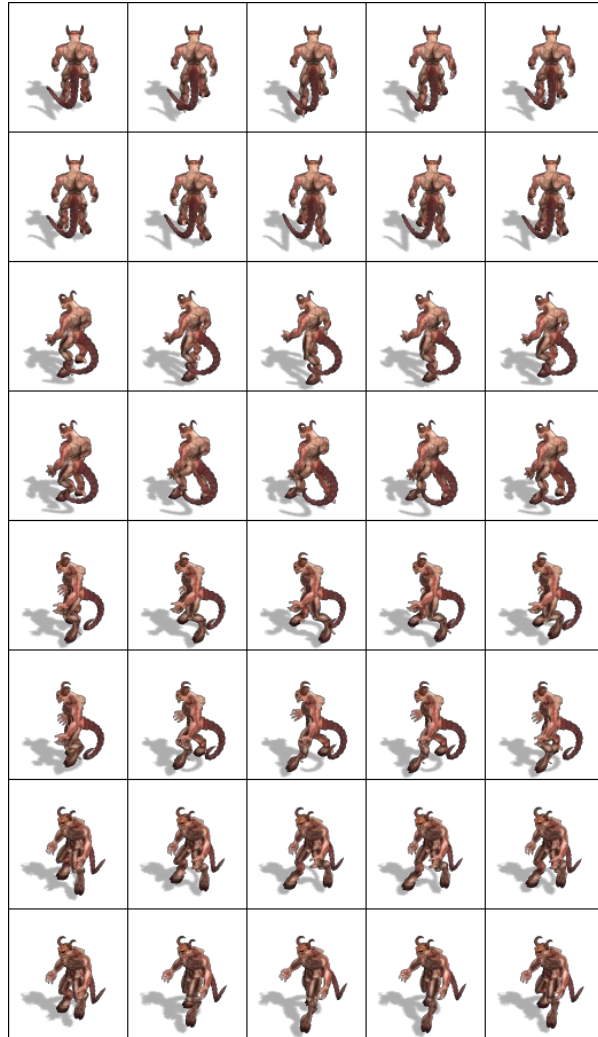


Drawing for Axonometric View

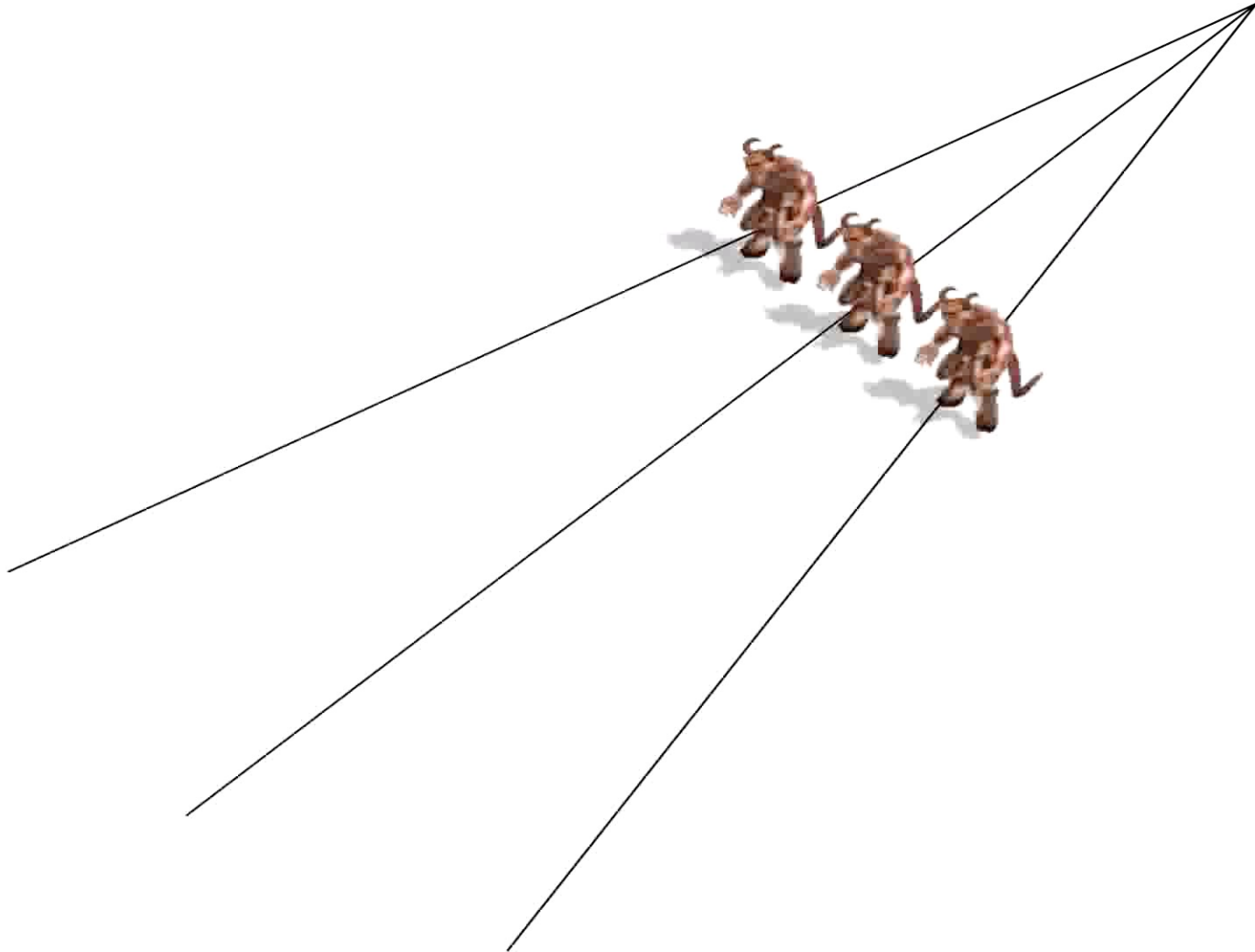
- Use boxes shown on slide
 - Tiling boxes is easy
 - Draw shape inside box
- Complex, large shapes?
 - Glue together boxes
 - Draw inside box group
- Objects need many angles
 - Transparency is tricky
 - Standard: 8 compass points
- **Example:** Terrain.ai



Isometric Walking Animation



Isometric Walking Animation



Isometric Animation: *Starcraft*



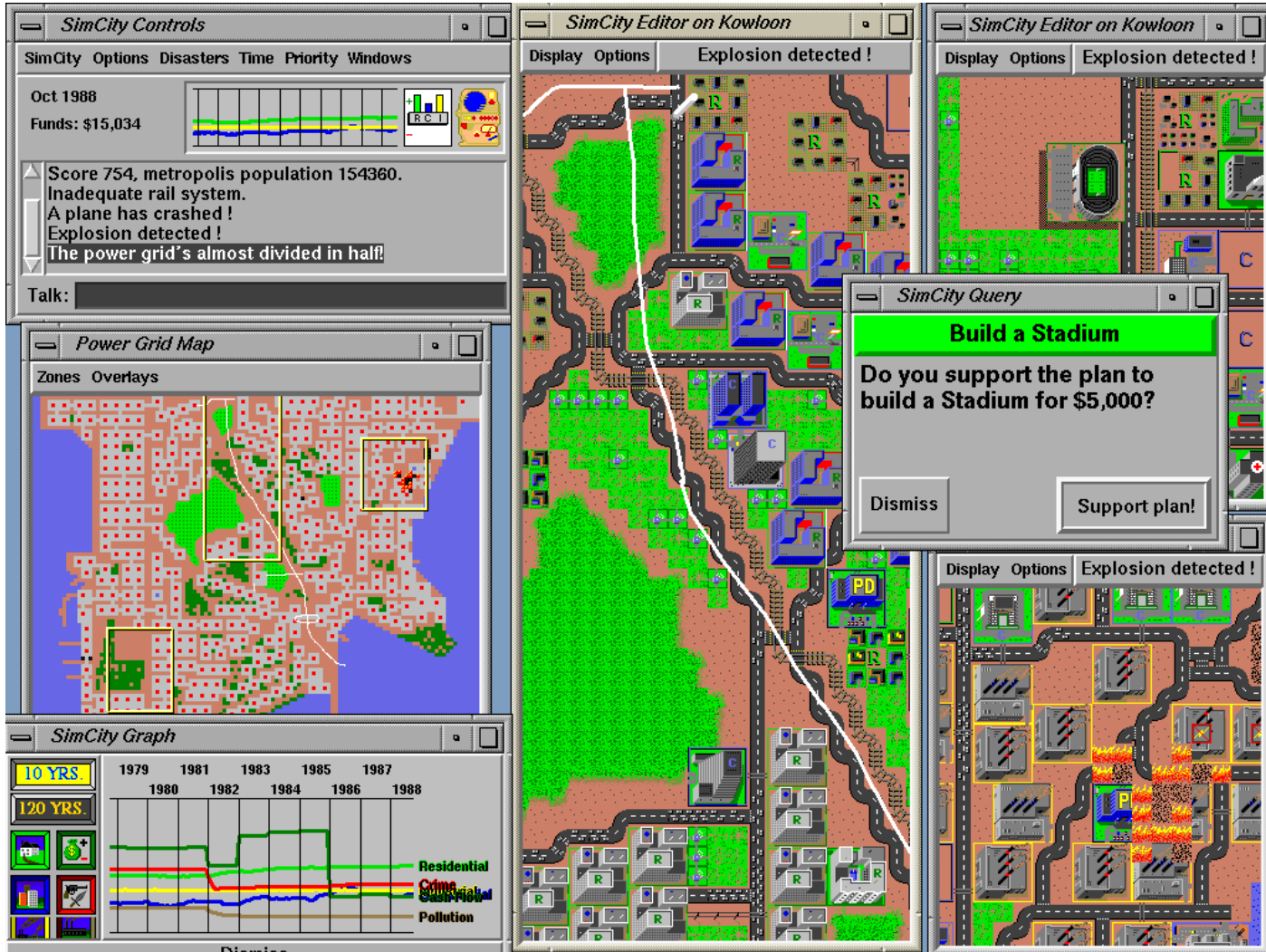
Oblique Perspective

- Less well-defined perspective category
 - Axonometric with “arbitrary” foreshortening
 - But game art is not always true mathematical
- But there are some historical categories
 - **Cabinet**: Used in cabinet maker drawings
 - **Military**: Used in classic military maps
 - See Wikipedia page for more details
- **In practice**: orthographic with slight *flair*

Cabinet Perspective: *Prince of Persia*



Military Perspective: *Sim City*



Which Style to Use?

Orthographic

- **Advantages**

- Easy to make tiles
- Easy to composite

- **Disadvantages**

- Movement is 2D
- Game feels flat

- Common in this class

Axonometric/Oblique

- **Advantages**

- Sort of easy to tile
- Some 3-D movement

- **Disadvantages**

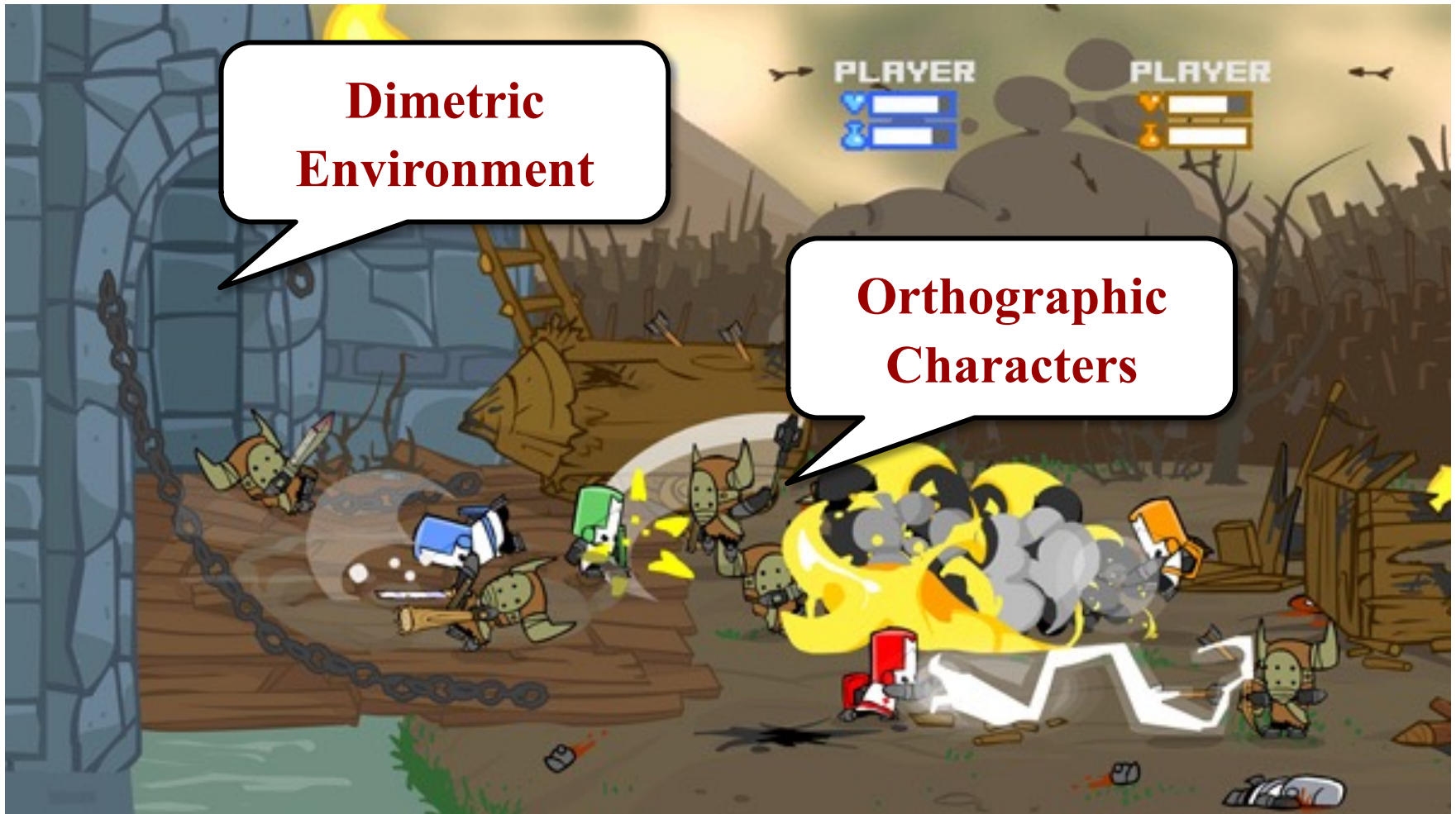
- Harder to composite
- Objects may be hidden

- Lot of work for artist

Combining the Perspectives



Combining the Perspectives



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