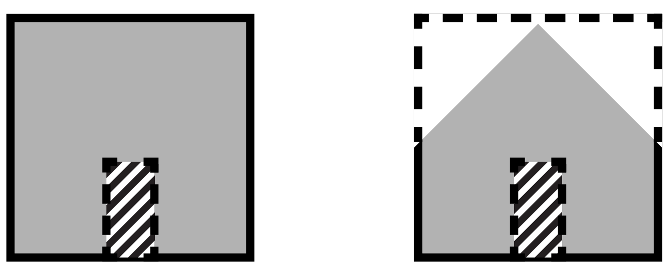
Rendering Basics

More Special Texture Mapping Tricks

- Aliasing concerns mean that you should never use textures with high internal contrasts
- Mipmapping
  - Store multiple smaller pre-antialiased texturemaps
  - Choose which texture to use based on depth
  - Takes 2x normal amount of texture memory (1+0.5+0.25+etc) = 2
- Add detail which is too complex for explicit polygonal modeling
  - facial features
  - text, body curves (as shades)
- Transparencies can model polygons



- Shadows
  - We can map pregenerated shadow maps from '3D sprites' onto the floor translucently, creating a perspective-correct shadow

Rendering Basics

Special Texture Mapping Tricks

- Reflections
  - Do another rendering using the reflecting polygon as the view plane
- Animation
  - Flip through textures (like animating sprites)
  - Add an offset to make textures rotate around objects (conveyor belts, etc)
- Light sourcing
  - Use prerendered radiosity (Gouraud shade squares in each texturemapped polygon)
  - We can incorporate Gouraud shading to represent reflected light from nearby light sources
- Depth Cueing
  - Helps the user distinguish far from near
  - With constant Z texture mapping we can simply apply a new shading value to each Z strip
  - No new loop needed for constant Z texture mapping
- Environment mapping
  - Take the *(x,y)* coordinates of the polygon vertex normals and use them as indicies into a circular texture map. Looks incredible as a metallic surface and can also simulate Phong shading if done with the correct texture.

Rendering Basics

Texture Mapping Approximations

- Arbitrary Angled Constant Z Mapping
  - Have to do bresenhams in screen space (as well as texture space) and draw along those lines
  - Much harder to implement
  - Pretty fast, but slower than affine or Doom
  - More aliasing artifacts than any other type
- Arbitrary 'Corrected' Affine Mapping
  - Corrected every 16x16 pixels
  - Linear in between these areas
  - Distortion is always small (but there is distortion)
  - No extra aliasing
  - Maybe it's faster than constant Z but who knows
  - Only about 10-20 people in the world know how hard it is to implement!
- Quadratic Texture Mapping
  - Uses a second degree polynomial to approximate the division curve needed to simulate perspective
  - Looks better than linear
  - Not much more inner loop computation than linear (just another addition using forward differencing)
  - More precomputation than linear
  - Still warps wierdly, but maybe not enough for you to care

Rendering Basics

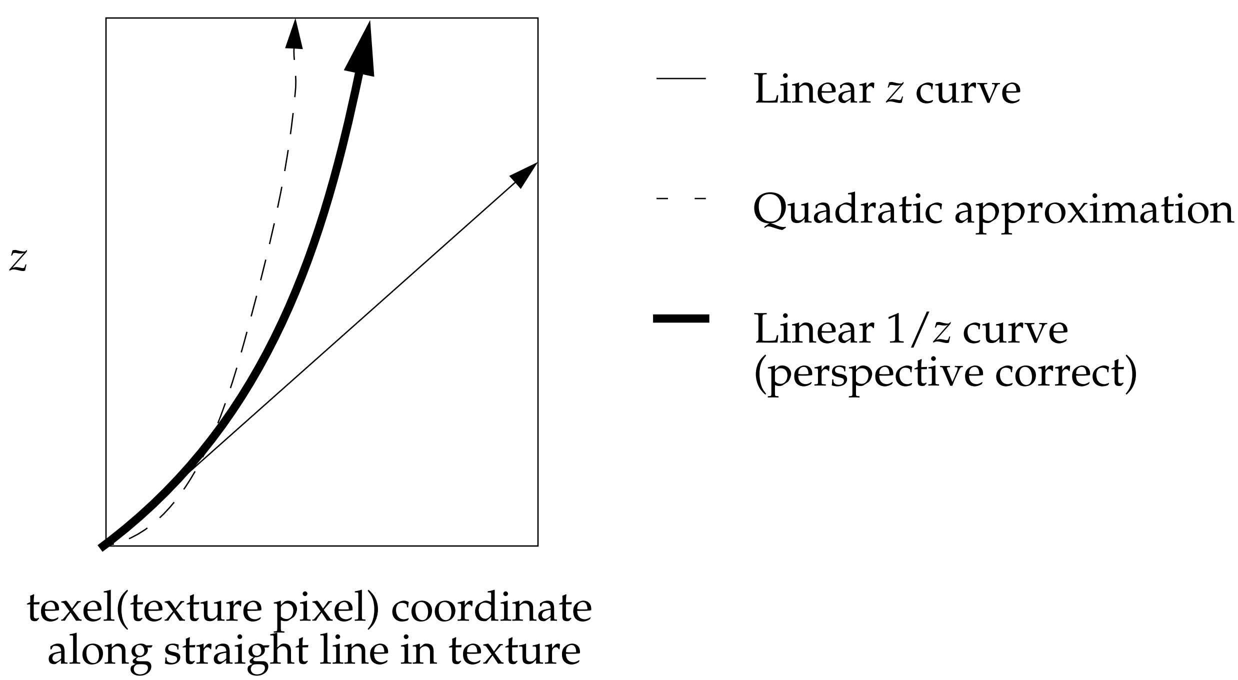
Texture Mapping Approximations

- Linear (affine) Texture Mapping
  - Find *u, v* coordinates at vertices
  - Step horizontally along polygon, linearly interpolating the *(u,v)* and looking up colors at each pixel
  - Linearly interpolate *(u,v)* in outer loop to get to the next scan line
  - Very fast
  - Easy to code
  - Unless you subdivide polygons, when they get large they will have a nasty curve inside their surfaces
- 2-Divisions per pixel method
  - Very slow
  - Perfectly correct (no curvies!)
  - Easy to code
- Constant Z Texture Mapping (DOOM)
  - Very fast (almost as fast as affine)
  - Perfectly correct
  - Easy to code
  - Doesn't work if the user looks up or down (much)
  - Can't draw surfaces which are not parallel with Y or with the XZ plane

Rendering Basics

3D Rendering

- Polygon scan conversion
  - Use triangles or **convex** quads
  - Go across horizontal lines scanning in color
  - Use bresenham or DDA to go down sides of triangle or quad
- Texture Mapping
  - We've covered 'perspective-correct' texture mapping in CS123, but it's SLOW!
  - The perspective equations are NOT linear
  - There are various hacks around these problems, and if you come up with a cool new one you'll probably become a millionaire



Spatial Coherency

An argument for object based rendering

- All scenes are made up of some kind(s) of primitive object(s) These are often polygons, but there are games which use ellipses and other shapes as well.
- Each primitive usually takes up many pixels on the display
- So, a good way to keep spatial coherency is to draw an object at a time. (In CS123 Ray we drew a pixel at a time)
- We can use a reverse-mapping in conjunction with 'forward differencing' to draw objects extremely quickly, one scanline at a time.

Drawbacks

- We can use the painters' algorithm, but then we sometimes have to draw pixels twice.
- Texture-mapping usually requires a non-linear computation (Perspective correction)

Differences from CS123 (3D)

SPEED SPEED SPEED SPEED SPEED!!!!

- We need each scene to render in 1/15th-1/70th of a second.
- RayTracer from CS123 took minutes to render a scene with only 5 objects in it.
- If possible, we need to avoid using any higher mathematical functions.
- We don't care about visual correctness. As long as the user can tell what is going on in the scene and it looks pretty, we've won.
- We may have to render into a paletted format, instead of RGB triples
- We never want to draw a pixel twice.
- We want to take advantage of coherency.
  - Spatial coherency
  - Temporal coherency
  - Cache coherency

Rendering Basics

2D Rendering

- Lines
  - Bresenham
  - DDA (a cheapo Bresenham)
- Sprites
  - Run Length Encoding (RLE)
  - Scaling
  - Rotation
  - Draw/Undraw cycle (Dirty rects. vs. full screen refresh)
  - Shading
- Backgrounds
  - Tiled
  - Parallax (RLE?)
  - Shading
- Isometric rendering concerns
  - Tile-based approaches
  - Z-buffers

Types of Graphics

Rendering Styles

- 2D games
  - Lines
  - Sprites
  - Tiled backgrounds
  - Parallax backgrounds
  - Palette manipulation for simulation of motion
  - Isometric sprite obscuring
- 3D games
  - ‘3D’ Sprites
  - Polygonal character models
  - Z-buffered static background models
  - 2.5D Environments
  - Fully polygonal world models
  - Voxel heightfield world models

Types of Graphics

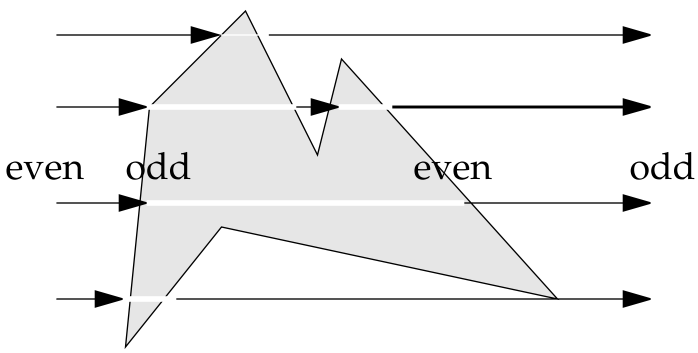
Standard gaming models

- 2D Puzzle Games
- 2D War Simulations
- Platform Games
- 2D Fighting Games
- Isometric ‘3D’ Games
- Alone-in-the-Dark Games (Z-buffered prerendered screens with polygonal characters superimposed)
- 3D Fighting Games (Virtua Fighter)
- Doom Games (First person perspective, fixed up/dn viewing angle)
- Commanche Games (Voxel heightfields)
- Descent Games (FPP, arbitrary angle)
- 3D Third Person Perspective Games (Mario Bros. 64)

Quick Overview of CS123

Algorithms

- Polygon Scan Conversion
  - Use bresenham to step along the edges of the polygon, one scanline at a time
  - Use the parity rule to fill in points along horizontal scanlines between edges



- Scaling
  - Step along source image in fractional increments
  - Draw to destination image {up to down {left to right}}
  - Get color to draw from anti-aliased section of area surrounding the current fractional position in the source image
  - Using a triangle filter, image expansion becomes computationally simple
- Rotation
  - Done exactly like scaling, but instead of going along the source in fractional horizontal increments, we step along a 2D bresenham line through the source image

Quick Overview of CS123

Algorithms

- Bresenham
  - 8 cases, one for each octant
  - Special speed-up cases for horizontal, vertical, and diagonal lines

```
/* Case 2 (octant 2 [dy>=0 && dy<=dx]) */
// This code isn't optimized!!!

// Setup variables
int deltax = x2 - x1;
int deltay = y2 - y1;
int y = y1;
int d = (2 * deltay) - deltax;

// Do inner loop
for(int x = x1; x < x2; x++) {
   DrawPixel(x, y, color);
   if(d < 0)
      d = d + (2 * deltay);
   else {
      d = d + 2 * (deltay - deltax);
      y++;
   }
}
```

Quick Overview of CS123

Basic Concepts

- Cumulative/Inverse Transforms
- Diffuse Lighting Model
  - The light contributed by a given light source is proportional to the cosine of the angle between the normal of a surface and the light vector.
  - This can be expressed as the dot product of the normal and light vectors.
  - Light is also proportional to the inverse square of the distance from the light source if we want to be physically accurate.
- Forward Differencing
  - One of the most important optimization concepts in graphics
  - States that instead of evaluating a given function at every point along a linear progression, we can simply add the derivative of this function to the previous result in order to obtain the next result
  - We can use this as an inductive hypothesis.
  - In other words, you can forward difference a forward difference. This generates correct results for functions with more than one non-zero derivative.
  - Works for any polynomial function
  - Accuracy is limited by the accuracy of your numerical representation

Quick Overview of CS123

Basic Concepts

- Viewing frustum (90 degrees looks nice!)
- All objects in a given Z plane appear the same size
- Perspective Projection
  - $x_{screen} = k_x + x_{worldspace} / z_{worldspace}$
  - $y_{screen} = k_y + y_{worldspace} / z_{worldspace}$
  - Straight lines are preserved after projection
- Rotations in 3-space
  - Rotate around x-axis

    (1) $y' = y \cos(t) - z \sin(t)$

    (2) $z' = y \sin(t) + z \cos(t)$
  - Rotate around y-axis

    (3) $x' = x \cos(t) + z \sin(t)$

    (4) $z' = -x \sin(t) + z \cos(t)$
  - Rotate around z-axis

    (5) $x' = x \cos(t) - y \sin(t)$

    (6) $y' = x \sin(t) + y \cos(t)$
- Aliasing/Anti-aliasing
  - Images can be decomposed into sine waves via Fourier.
  - High frequencies must be culled when compressing images to avoid random patterns.
  - Low frequencies must be accurately represented when expanding images.

Introduction to Realtime Graphics Rendering

Daniel Goldstein
Samuel Trychin
Douglas Applewhite