# Textured Dreams

Nicholas Markle

CS 4204 Final Project

# What did I do

MY FINAL PROJECT WAS TO IMPLEMENT UVS INTO THE EXISTING GRAPHICS PIPELINE

UVS ARE 2D COORDINATES USED TO MAP A 3D MODEL'S SURFACE TO A 2D TEXTURE

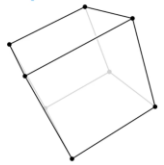
DEFINE HOW TEXTURES ARE WRAPPED AROUND THE MODEL.

LEARNED HOW TO IMPLEMENT FROM THE LECTURE SLIDES AND KEENAN CRANE’S VIDEO.

# What are UVs

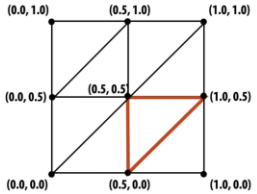
## Texture coordinates

example: texture this cube

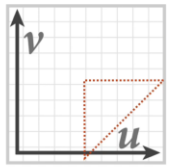


- "Texture coordinates" define a mapping from surface coordinates to points in texture domain
- Often defined by linearly interpolating texture coordinates at triangle vertices

Suppose each cube face is split into eight triangles, with texture coordinates (u,v) at each vertex

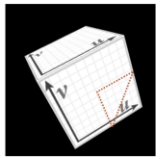



A texture on the $[0,1]^2$ domain can be specified by a 2048x2048 image




(location of highlighted triangle in texture space shown in red)

Linearly interpolating texture coordinates & "looking up" color in texture gives this image:




CMU 15-462/662

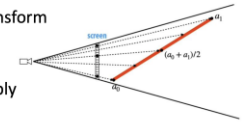
## Texture Perspective

Project Point (perspective)

Due to perspective projection (homogeneous divide), barycentric interpolation of values on a triangle with different depths is not an affine function of screen XY coordinates

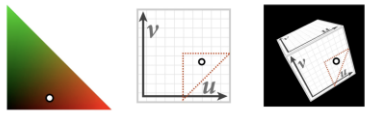
1. Apply Inverse of Camera Transform
2. Apply ***P***
3. Divide by last element to apply perspective scaling
4. Apply ***O***

Homogenous Coordinate (W)




## Texture Sampling 101

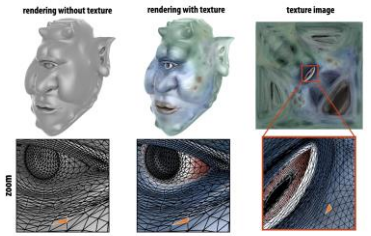
- Basic algorithm for texture mapping:
  - `for each pixel in the rasterized image:`
    - `interpolate` $(u, v)$ `coordinates across triangle`
    - `sample (evaluate) texture at interpolated` $(u, v)$
    - `set color of fragment to sampled texture value`




...sadly not this easy in general!

CMU 15-462/662

## Texture mapping adds detail




CMU 15-462/662

## Texture Perspective

- Goal: interpolate some attribute $\phi$ at vertices
- Basic recipe:
  - Compute depth z at each vertex
  - Evaluate Z := 1/z and P := $\phi$/z at each vertex — U, V coords / z
  - Interpolate Z and P using standard (2D) barycentric coords
  - At each *fragment*, divide interpolated P by interpolated Z to get final value — z during inverse project point == w

$$w = \frac{n \cdot f}{f + n - p''.z}$$

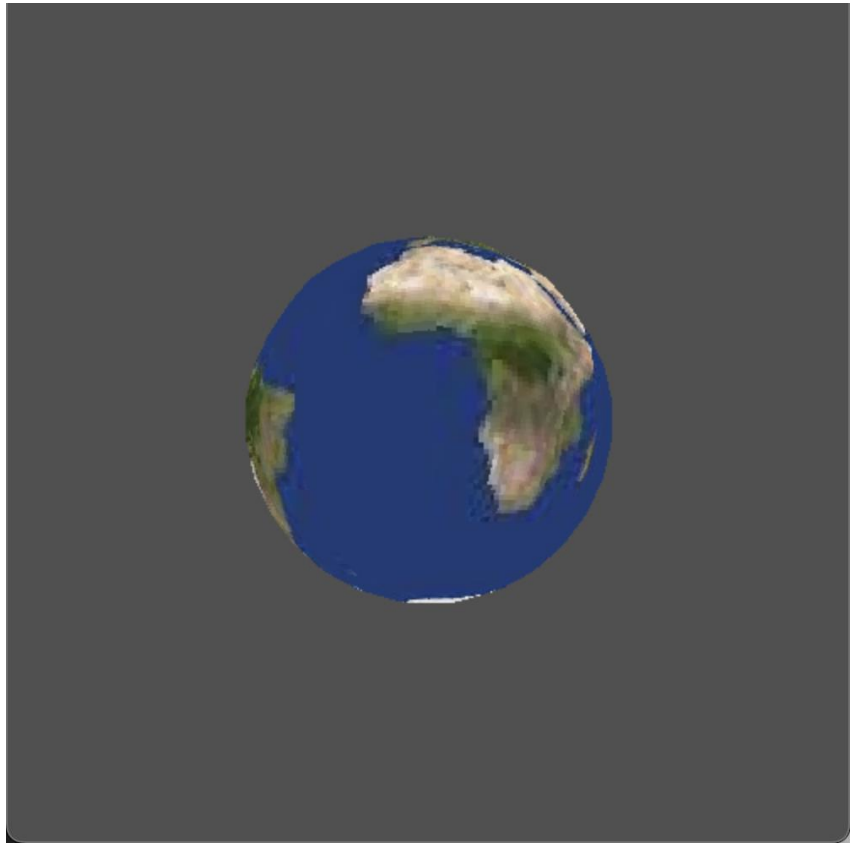
a derivation, see Low, "Perspective-Correct Interpolation"

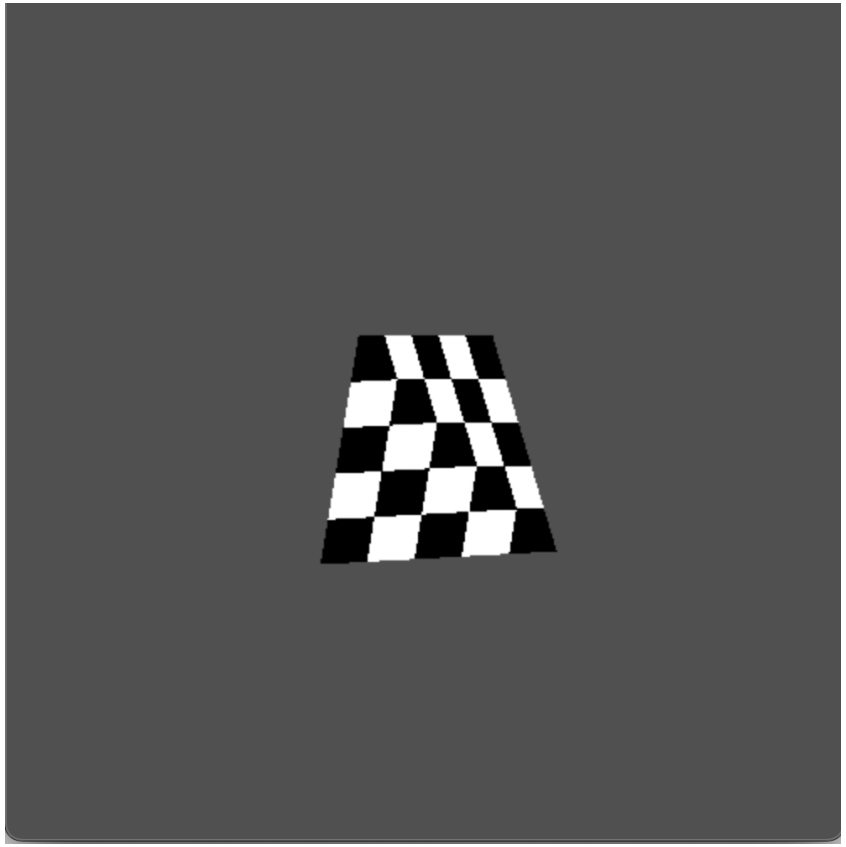
CMU 15-462/662

# Affine implementation

- **Barycentric Coordinates:** Calculate the texture coordinates (u and v) for the pixel using barycentric interpolation (alpha, beta, gamma) based on the triangle's vertices.
- **Clamping:** Ensure u and v are within the valid texture bounds (0 to 1)
  **Texture Sampling:** Sample the texture at the computed coordinates (u, v) and assign the resulting color to the image buffer.
- **Depth:** Update the depth buffer with the interpolated depth value (z_interpolated).

# Output from running Affine Implementation

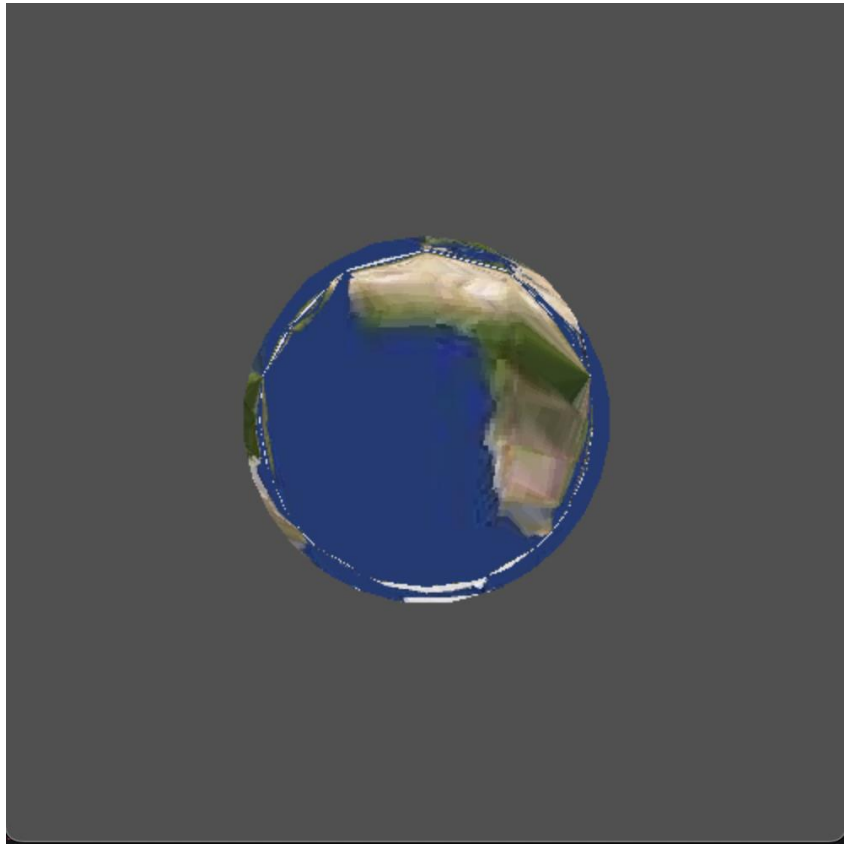




# Perspective Correct Implementation

- **Inverse Depths:** Calculate the inverse of the depth values (1/z0, 1/z1, 1/z2) for each triangle vertex.
- **Depth Correction:** Adjust the barycentric coordinates (alpha, beta, gamma) using the inverse depth values to handle perspective correction (alpha_w, beta_w, gamma_w).
- **Normalization:** Normalize the corrected weights (alpha_w, beta_w, gamma_w).
- **Perspective-Correct UV Calculation:** Compute the corrected texture coordinates (u, v) by weighting the UVs of each vertex based on the perspective-corrected barycentric coordinates.
- **Clamping:** Ensure the corrected texture coordinates are within valid bounds (0 to 1).
- **Texture Sampling:** Sample the texture at the corrected coordinates and assign the resulting color to the image buffer.
- **Depth:** Update the depth buffer with the interpolated depth value (z_interpolated).

# Didn’t quite get this working



# Final Remarks

Affine Implementation worked well

Perspective Correct Implementation is a bit buggy

Wish I hadn’t been sick so much this semester