Lecture 4

Implicit and Parametric Representations
Triangle Meshes: Texture Coordinates
Big Math Idea:

Implicit vs Parametric Representations

*Implicit*: a *property* that's *true* at all points

*Parametric*: a *recipe* for *generating* all points
Implicit vs Parametric: Lines

Implicit:

\[ y = mx + b \]
\[ y = 2x + 1 \]
\[ 3 = 2 \cdot 1 + 1 \checkmark \]

\((x, y)\) is on the line if \( y = mx + b \)

Alternatively: \( ax + by + c = 0 \)
\[ y = 2x + 1 \]
\[ -2x + y - 1 = 0 \]
\[ \frac{a}{b} < \]

\[ 0x + y - 4 = 0 \]
\[ x = 1 \]
\[ 1x + 0y - 1 = 0 \]
Implicit vs Parametric: Lines

**Parametric:**

\[
\begin{align*}
X &= (0) + (1) \cdot t \\
y &= (1) + (2) \cdot t
\end{align*}
\]

Pick any \( t \), \((x, y)(t)\) lies on the line.

Alternatively:

\[
\begin{bmatrix} 0 \\ 1 \end{bmatrix} + t \begin{bmatrix} 1 \\ 2 \end{bmatrix}
\]

Works in 3D too!
Parametric Surfaces

• Sometimes it's useful to have 2D coordinates for positions on a 3D surface.
• This is called parameterizing the surface.
• Examples:
  • Cartesian coordinates on a 3D plane
  • Latitude and longitude on Earth's surface
  • Spherical coordinates ($\theta$, $\phi$) on a sphere
  • Cylindrical coordinates ($\theta$, $y$) on a cylinder
Example: Earth

Two coordinates (lat, lon) identify a position in 3D space.

This is possible because the earth is a 2D surface (manifold)
Implicit vs Parametric: Planes

Parametric:
\[ \mathbf{\rho} + s \mathbf{u} + t \mathbf{v} \]

Implicit:
\[ \mathbf{n} \cdot (\mathbf{x} - \mathbf{\rho}) = 0 \]
Implicit vs Parametric: Sphere

Assume: unit radius centered at \((0,0,0)\)

**Implicit:**

\[ x^2 + y^2 + z^2 = 1^2 \]
\[ x^2 + y^2 + z^2 - 1 = 0 \]

**Parametric:**

\[ x = r \cos \phi \sin \theta \]
\[ y = r \sin \phi \sin \theta \]
\[ z = r \cos \theta \]
Last time: data on Meshes

- Often we need more than just geometry.
- Many properties vary continuously over a smooth surface.
Data on Meshes

• What do we need to store at vertices?

  • **Surface Normals**
    to more accurately portray geometry

  • **Texture Coordinates**
    to paste image data onto surfaces

• **Positions!**?
  just another piece of per-vertex data!
Textures

You are here:

You wish to be here:

Using current machinery: store a color at each vertex and interpolate between them.

We'd need a bunch more triangles.
Textures

You are here: You wish to be here:

We'd need a bunch more triangles.
Textures

• Store **spatially varying surface properties**:
  
  • color is an intuitive example, but many other things too; anything that changes over the surface but doesn't affect geometry (much)
  
  • roughness, faked lighting effects, normals(!?), bumps
What is a texture?

- A texture is basically a 2D image that stores some spatially-varying surface property.

  (use color for intuition, but keep in mind it's more general)

2D grid of values ("texels")
u, v coordinates in [0, 1]
Texture Mapping

• To use this, we need a **mapping** (function)
  • from the surface we're modeling/rendering
  • to \((u,v)\) **texture coordinates**

• **Simplest possible example:**
  a 2x2 tabletop in the xz plane

• When rendering, non-vertex points get colors via **interpolated** \((u,v)\) coordinates.
Texture Mapping Function

\( \phi(x, y, z) \Rightarrow u, v \)
Modeling the Tabletop

Let's write an OBJ file

(1) v 0 0 0
(2) v 0 0 2
(3) v 2 0 2
(4) v 2 0 0

t 0 0
t 1 0
t 1 1
t 0 1

f 1/1 2/2 3/3
f 1/1 3/3 4/4

eral: pos/ tex/ normal

Recall: p/t p/n p/4/n
Texture Mapping: nontrivial surfaces

Map from point on sphere to point in (u,v)
A1 sphere - demo

- North pole \( v = 1 \)
- 2 texture coords at each location on seam
- 32 different texture coordinates at pole (leave 1 out)
- 17 vertices spaced uniformly pole to pole
- every other triangle around pole is missing
- each rectangle of the latitude-longitude grid represented by 2 triangles
- 32 vertices \( v = 0.5 \) spaced uniformly around equator
- \( u < 0.5 \) when \( x < 0 \)
- \( u = 0.5 \) when \( x = 0 \)
- \( u > 0.5 \) when \( x > 0 \)
- \( v < 0.5 \) when \( y < 0 \)
- \( v = 0.5 \) when \( y > 0 \)
- \( u = 0.75 \)
- \( v = 0.5 \)
Texturing the Pyramid: The Texture

Textures aren't necessarily square - still [0, 1]
Texturing the Pyramid: The Texture Mapping Function

- **Base**
- **Sides**
- **(apex)**

- Points:
  - $(0, 0)$
  - $(0, 0.5)$
  - $(1, 0.5)$
  - $(1, 1)$

The diagram illustrates the texture mapping function for a pyramid, with coordinates $(u, v)$ mapping to the geometric features of the pyramid.
Texturing the Pyramid: The Texture Mapping Function