Depth and Shadows

Dinesh K. Pai

Textbook Chapter 11

Several slides courtesy of M. Kim

1

Today

- Announcement: A4 will be available later today
  - Demo
- Preparation for Quiz 3
- Depth and Shadows

2

Quiz 3 Preparation

- In class, Friday March 27 1-1:50. Please be on time.
- Review lecture notes, and assignments.
- Everything covered in lecture could be on the exam
- Everything covered in listed textbook chapters could be on the exam
- Doing first part of Assignment 4 will be very helpful

3

Quiz 3 Preparation

- Textbook. Read ALL of these, except as noted
  - Ch 15 Texture Mapping. Main focus of quiz
  - Ch 16 and 17, portions that will be covered Monday/Wednesday
  - Interpolation: focus on L20,L21, esp. Bernstein polynomials, but skim Ch 9
  - Projection. Mainly focus on recent lectures on use with projector textures. Skim Ch 10, 11.3
  - Ch 11 Depth
  - Ch 12 From Vertex to Pixel. Only Section 12.3.
- Topics from Quiz 1 and 2 will be assumed as pre-requisites (e.g., it is assumed you now know coordinate frames and how to transform them)

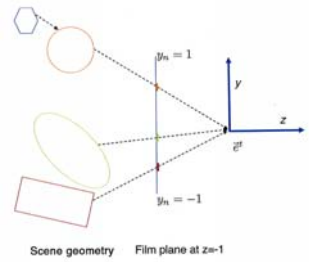
4

Depth Demo

- http://threejs.org/docs/#Reference/Materials/MeshDepthMaterial

5

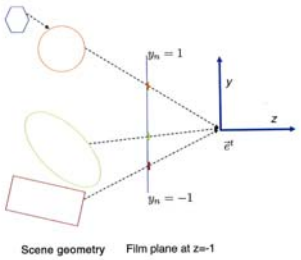
Visibility



- In the real world, opaque objects block light.
- We need to model this computationally.
- One idea is to render back to front and use overwriting
  - This will have problem with visibility cycles.

6

Visibility



- We could explicitly store everything hit along a ray and then compute the closest.
  - Make sense in a ray tracing setting, where we are working one pixel per ray at time, but not for OpenGL, where we are working one triangle at a time.

7

Z-buffer

- We will use z-buffer (or depth buffer)
- Triangles are drawn in any order
- Each pixel in frame buffer stores 'depth' value of closest geometry observed so far.
- When a new triangle tries to set the color of a pixel, we first compare its depth to the value stored in the z-buffer.
- Only if the observed point in this triangle is closer, we overwrite the color and depth values of this pixel.

8

Z-buffer

- This is done per-pixel, so there is no cycle problem.
- There are optimizations, where z-testing is done before the fragment shading is done.

9

Other uses of visibility calculations

- Visibility to a light source is useful for shadows.
  - We will talk about shadow mapping later.
- Visibility computation can also be used to speed up the rendering process.
  - If we know that some object is occluded from the camera, then we don't have to render the object in the first place.
  - We can use a conservative test.

10

Basic mathematical model

- For every point, we define its $[x_n, y_n, z_n]^t$ coordinates, using the following matrix expression:

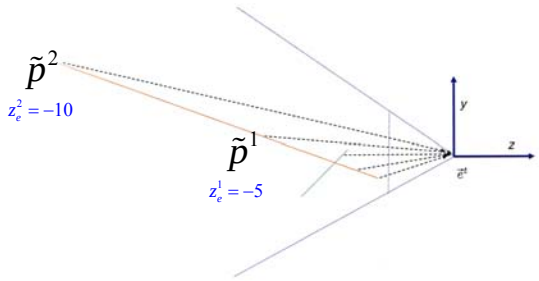
$$\begin{bmatrix} x_n w_n \\ y_n w_n \\ z_n w_n \\ w_n \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} s_x & 0 & -c_x & 0 \\ 0 & s_y & -c_y & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}$$

- We now also have the value $z_n = \frac{-1}{z_e}$
- Our plan is to use this z_n value to do depth comparison in our z-buffer.

11

Correct ordering

- Given two points $\tilde{p}^1$ and $\tilde{p}^2$ with eye coordinates $[x_e^1, y_e^1, z_e^1, 1]^t$ and $[x_e^2, y_e^2, z_e^2, 1]^t$.
- Suppose that they both are in front of the eye, i.e., $z_e^1 < 0$ and $z_e^2 < 0$.
- And suppose that $\tilde{p}^1$ is closer to the eye than $\tilde{p}^2$, that is $z_e^2 < z_e^1$
- Then $-\frac{1}{z_e^2} < -\frac{1}{z_e^1}$, meaning $z_e^2 < z_e^1$



Projective transform

- We can now think of the process of taking points (given by eye coordinates) to points (given by normalized device coordinates) as an honest-to-goodness 3D geometric transformation.
- This kind of transformation is generally neither linear nor affine, but is something called a *3D projective transformation.*
- Projective transformation preserves co-linearity and co-planarity of points.

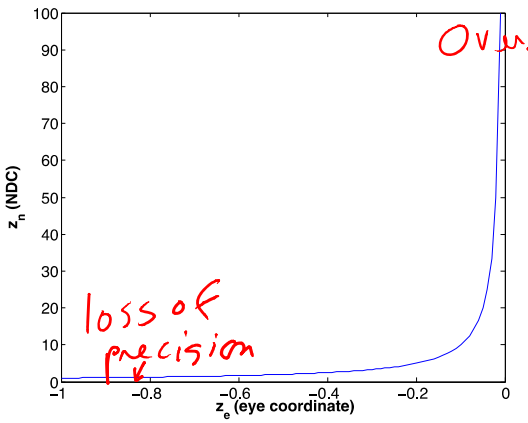
13

Numerics

- points very far from the eye have z_n values very close to zero

$$z_n = \frac{-1}{z_e}$$

```
ze=-1*[0.01:0.01:10];
zn=-1./ze;
plot(ze(1:100),zn(1:100))
```



Choose near plane carefully

14

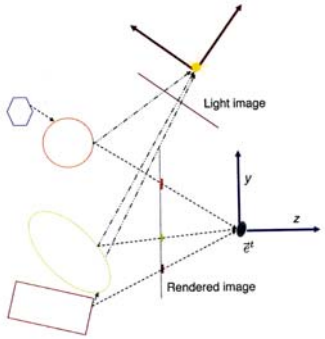
How to use it

- In Three.js, depth tests are on by default. See THREE.Material (.depthTest and .depthWrite)
- In OpenGL/WebGL, the z-buffer is turned on with a call to glEnable(GL_DEPTH_TEST).
- We may also need a call to glDepthFunc(GL_GREATER), since we are using a right handed coordinate system where 'more-negative' is 'farther from the eye'.

15

Shadow mapping

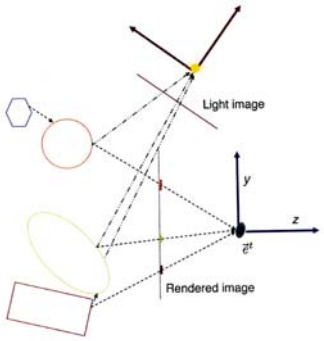
- First pass: create "shadow map", a z-buffer image from the point of view of the light
- Second pass: check if fragment is visible to the light using shadow map.



16

Shadow mapping

- If a point observed by the eye is not observed by the light, then there must be some occluding object in between, and we should draw that point as if it were in shadow.



17

Shadow mapping

- In a first pass, we render into an FBO the scene as observed from some camera whose origin coincides with the position of the point light source. Let us model this camera transform as:

$$\begin{bmatrix} x_t w_t \\ y_t w_t \\ z_t w_t \\ w_t \end{bmatrix} = P_s M_s \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

for appropriate matrices, P_s and M_s .

18

Shadow mapping

- During this first pass, we render the scene to an FBO using M_s as the modelview matrix and P_s as the projection matrix.
- In the FBO we store, not the color of the point, but rather its "depth value".
- Due to z-buffering, the data stored at a pixel in the FBO (depth value), is a monotone function of z_t. This FBO is then transferred to a texture.

19

Shadow mapping

- During the second rendering pass, we render our desired image from the eye's point of view, but for each pixel, we check and see if the point we are observing was also observed by the light, or if it was blocked by something closer in the light's view.
- To do this, we use the same computation that was done with projector texture mapping
- Doing so, in the fragment shader, we can obtain the varying variables x_t, y_t and z_t associated with the point $[x_o, y_o, z_o, 1]^t$.

20