CPSC 314 Computer Graphics

Dinesh K. Pai

Texture Mapping 4: Visibility, Depth, Shadows

Chapter 11,15

Some slides courtesy of M. Kim, KAIST

Preliminaries

- Announcements and Reminders
  - Plans for next week
- Today
  - Assignment 2 Spotlights
  - Depth and shadows

Plans for next week

- March 18 Quiz 2, same location as Midterm
- March 17 no office hour
  - I'll be returning from a conference late 17th
  - Please post any Quiz-related questions on Piazza and tag as "quizzes" and I'll respond asap
- March 16 Industry Guest Lecture #1, on Zoom
- March 14 Review class

3

SunTracker Technologies Ltd.

Computer Graphics

Looking Beyond Games

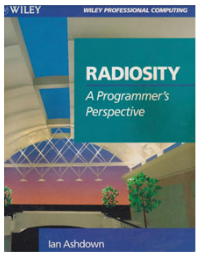
CPSC 314 special guest

Ian Ashdown, P. Eng. (Ret.), FIES
Senior Scientist
SunTracker Technologies Ltd.

SunTracker Technologies Ltd.

Presenter – Ian Ashdown

- UBC Computer Science graduate (2001)
  - *Eigenvector Radiosity* (MSc thesis)
  - *Radiosity: A Programmer's Perspective* (Wiley 1994)
- Software Engineer
  - Architectural lighting design and analysis
  - 40,000+ clients worldwide
- Solid-state Lighting Research Engineer
  - 150+ patents and patent applications
- Architectural / Horticultural Lighting Researcher
  - 160 papers and articles



A2 spotlights

- Anna Wang
- Anonymous (name withheld upon request)
- Nick Zhang
- Peter Newman

6

Shadows

Dinesh K. Pai

Textbook Chapter 11

Several slides courtesy of M. Kim

7

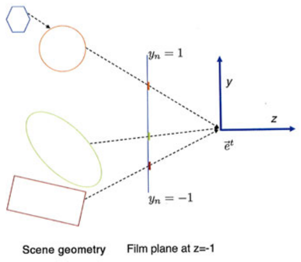
Recap Lecture on Depth

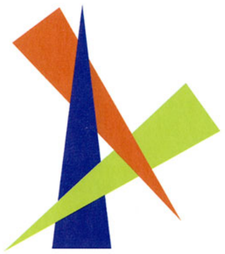
Depth Demo

https://threejs.org/docs/#api/materials/MeshDepthMaterial

8

Visibility





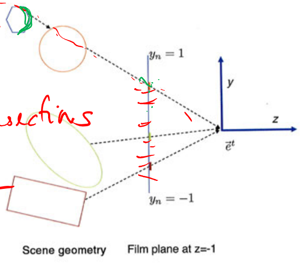
- In the real world, opaque objects block light.
- We need to model this computationally.
- One idea is to render back to front and use overwriting
  - This will have problem with visibility cycles.

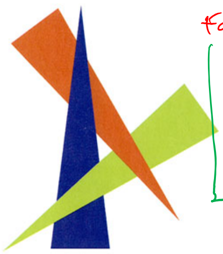
9

Visibility

Ray casting

```
For all screen pixels
    For all objects
        Compute intersections
    Return closest point
```





Depth Buffer

```
For all objects
    For all fragments (pixels)
        Record depth
Return closest
```

- 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.

10

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.

11

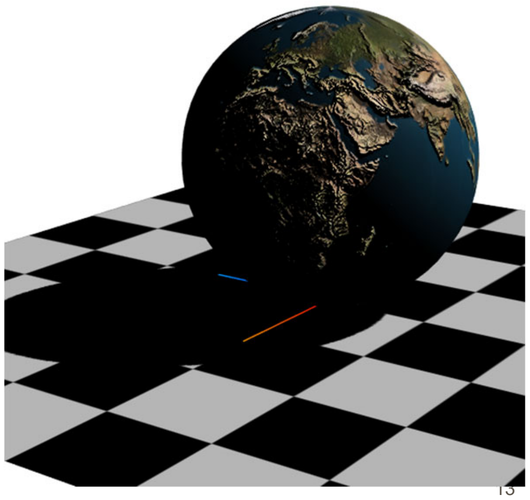
Depth Demo

https://threejs.org/docs/#api/materials/MeshDepthMaterial

12

Shadow Mapping

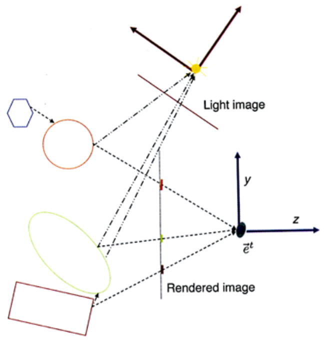
- Interactive example: helloWorld3

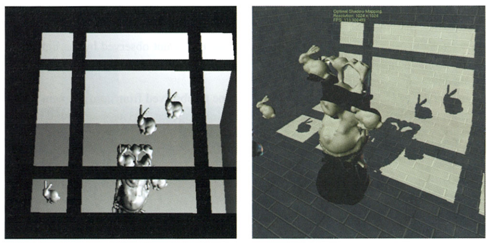


13

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.

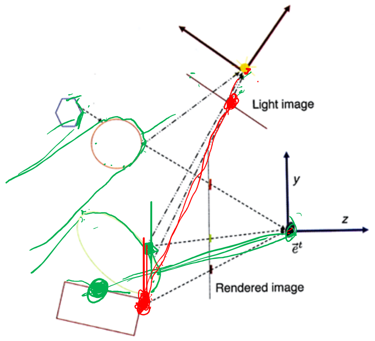


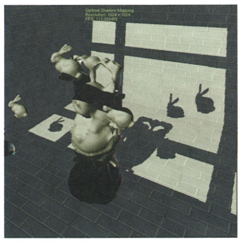


14

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.





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 .

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.

17

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.

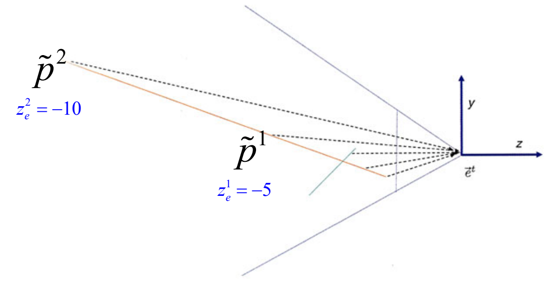
18

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

  $z_n^2 < z_n^1$



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.

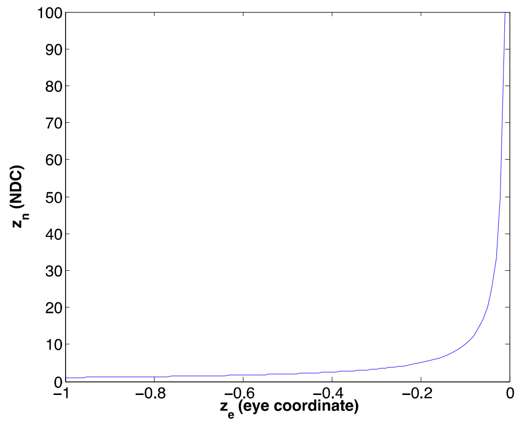
20

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))
```



21