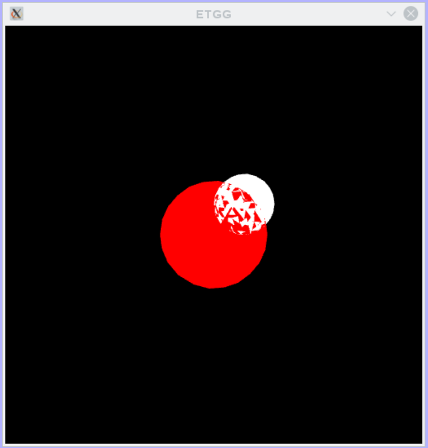
Depth Buffer

Motivation

- Overlapping triangles are not drawn correctly.

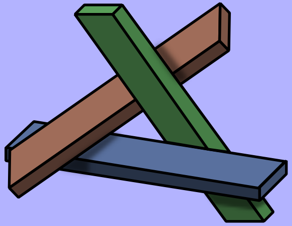


Overlapping Objects

- How to handle?
  - Could be several different objects that are overlapping
  - Could be several faces of one object that overlap
- Solutions:
  - About 10 developed in late 1970's-early '80's
  - Only three remain in wide use:
    - Painter's algorithm
    - Raytracing
    - Z buffer

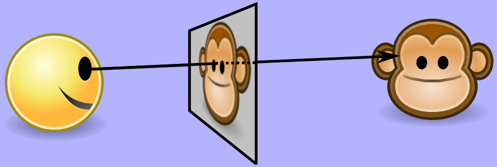
Painter’s Algorithm 

- Draw back to front
  - Simple & easy to understand
  - Not practical once scenes have more than a few items
    - Depth sorting becomes too expensive
  - Some objects can’t be rendered



Raytracing

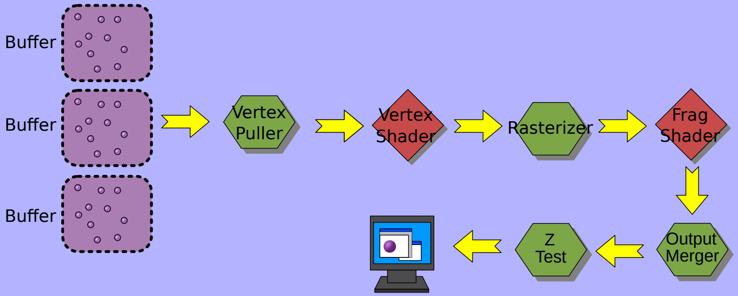
- Raytracing: Invented in $\approx$1983, but it's gaining new interest
- Real-time raytracing is *almost* mainstream



Depth Buffer

- Depth buffer (z buffer): Preferred strategy for current systems
- Idea:
  - For each pixel that we want to draw, determine what its depth is
  - See what the depth is of the point on the screen at that location
  - If the incoming pixel is closer than what's on the screen: Replace
  - Else, leave the existing pixel alone
- Simple and easy to parallelize $\rightarrow$ Common to have hardware acceleration

Architecture



Example

- Example rendering + Depth buffer

Depth Buffer

- Z buffer uses output of VS to do its work
- Recall our projection matrix: $\begin{bmatrix} \frac{1}{tan\,\theta_h} & 0 & 0 & 0 \\ 0 & \frac{1}{tan\,\theta_v} & 0 & 0 \\ 0 & 0 & -\left(1+\frac{2Y}{H-Y}\right) & 1 \\ 0 & 0 & \frac{2HY}{H-Y} & 0 \end{bmatrix}$
- When we compute p' = p M, we get

$$p' = [\ldots, \ldots, -\left(1+\frac{2Y}{H-Y}\right)z + \frac{2HY}{H-Y}, z]$$

Depth Buffer

- And after homogeneous divide:

$$p' = \left[\ldots, \ldots, -\left(1 + \frac{2Y}{H-Y}\right) + \frac{2HY}{(H-Y)z}, 1\right]$$

- This maps z to range -1...1 when it ranges from hither ... yon
- But notice it's *not* a linear graph!
  - Equation form: $f(z) = c_1 + \frac{c_2}{z}$

Depth Buffer

- Example: https://www.ssucet.org/~jhudson/18/2801/zgraph/

Precision

- Often, GPU uses 16 or 24 bit integer to store depth
  - Faster than using floating point
  - Less RAM
- Consider if we use hither=0.001, yon=1000: $z' \in [-1, 1]$:

$$z' = \frac{1000.001}{999.999} - \frac{2}{999.999z} \approx 1.000002 - \frac{0.002}{z}$$

  - To map z' to 16 bits: $z_m = \frac{z'+1}{2} \cdot (2^{16} - 1)$
  - To map z' to 24 bits: Multiply by $2^{24} - 1$ instead

Table

- Hither = 0.001, yon = 1000

p_z	p'_z	16-bit	24-bit
0.001	-1	0	0
0.002	0	32767	8388615
0.005	0.6	52428	13421785
0.01	0.8	58981	15099508
0.1	0.98	64879	16609459
1	0.998	65469	16760454
5	0.9996	65521	16773876
10	0.9998	65528	16775554
50	0.99996	65533	16776896
75	0.99998	65534	16777008
100	0.99998	65534	16777064
250	0.99999	65534	16777164
500	0.999998	65534	16777198
750	0.999999	65534	16777209
1000	1	65535	16777215

Observe

- If we have 16-bit depth buffer and two objects at 100 units and 700 units from the eye: System can't tell which is further!
  - In fact, about half of our precision is used for the range from hither...2*hither units from the eye
- For 24 bit z buffer, we only have 6 units for entire distance from 750 to 1000 units, so there will be *depth fighting* there as well

Solutions

- How can we address this? A few ways...
- Option 1: Try to make hither as far as possible and yon as close as possible
  - Hither has more effect on precision than yon
- Option 2: Request more precise z-buffer
  - 24 bit better than 16 bit
  - Some platforms allow 32 bit: Better yet
  - Some systems allow floating point z-buffer
- Option 3: Shader tricks
  - We won't discuss these now...

Yon

- Surprisingly, we can set yon *very* large and it gives workable results
- What if yon was infinite?
- Use calculus:

$$\lim_{Y\to\infty}\begin{bmatrix} \frac{1}{tan\,\theta_h} & 0 & 0 & 0 \\ 0 & \frac{1}{tan\,\theta_v} & 0 & 0 \\ 0 & 0 & -\left(1+\frac{2Y}{H-Y}\right) & 1 \\ 0 & 0 & \frac{2HY}{H-Y} & 0 \end{bmatrix}$$

Solve

- Use L'Hôpital's rule:

$$\lim_{Y\to\infty} -\left(1+\frac{2Y}{H-Y}\right) = -\lim_{Y\to\infty}\frac{H-Y+2Y}{H-Y} = -\lim_{Y\to\infty}\frac{H+Y}{H-Y} = -\lim_{Y\to\infty}\frac{1}{-1} = 1$$

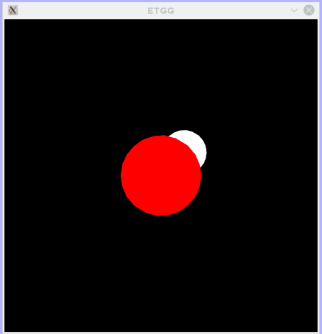
$$\lim_{Y\to\infty}\frac{2HY}{(H-Y)} = \lim_{Y\to\infty}\frac{2H}{-1} = -2H$$

Usage

- Using the depth buffer is simplicity itself:
- At setup time, enable depth test:
  - glEnable(GL_DEPTH_TEST)
  - glDepthFunc(GL_LEQUAL)
- When clearing screen, need to also clear depth buffer: glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

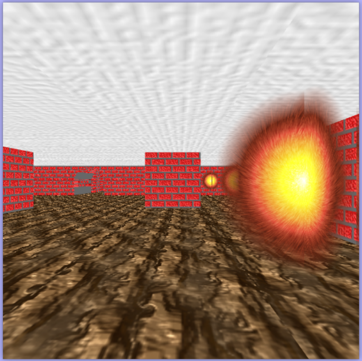
Result

- Correct display now



Assignment

- Add depth buffering to your previous lab
- Add the bullets back into your lab (fire a bullet when space is pressed)
  - You can start them at the camera's eye point, if you like
  - And then have them move in the direction the camera was facing when they were launched
  - If you have no artistic skills whatsoever, you can use this cheesy bullet mesh



Bibliography

- David Brackeen. *Game Programming in Java*. New Riders Media.
- D. Hearn & M. P. Baker. *Computer Graphics in C*.
- Eric Lengyel. Mathematics for 3D Game Programming & Computer Graphics (2nd ed.). Charles River Media.
- Frank Luna. *Introduction to 3D Game Programming with DirectX 9.0.* Wordware Publishing.
- Frank Luna. *Introduction to 3D Game Programming with DirectX 10.* Wordware Publishing.
- Khronos Group. WebGL Reference Card. http://www.khronos.org/webgl
- Brano Kemen. Outerra: Logarithmic Depth Buffer Optimizations & Fixes. http://outerra.blogspot.com/2013/07/logarithmic-depth-buffer-optimizations.html

Created using LaTeX.

Main font: Gentium Book Basic, by Victor Gaultney. See http://software.sil.org/gentium/
Monospace font: Source Code Pro, by Paul D. Hunt. See https://fonts.google.com/specimen/Source+Code+Pro and http://sourceforge.net/adobe
Icons by Ulisse Perusin, Steven Garrity, Lapo Calamandrei, Ryan Collier, Rodney Dawes, Andreas Nilsson, Tuomas Kuosmanen, Garrett LeSage, and Jakub Steiner. See http://tango-project.org