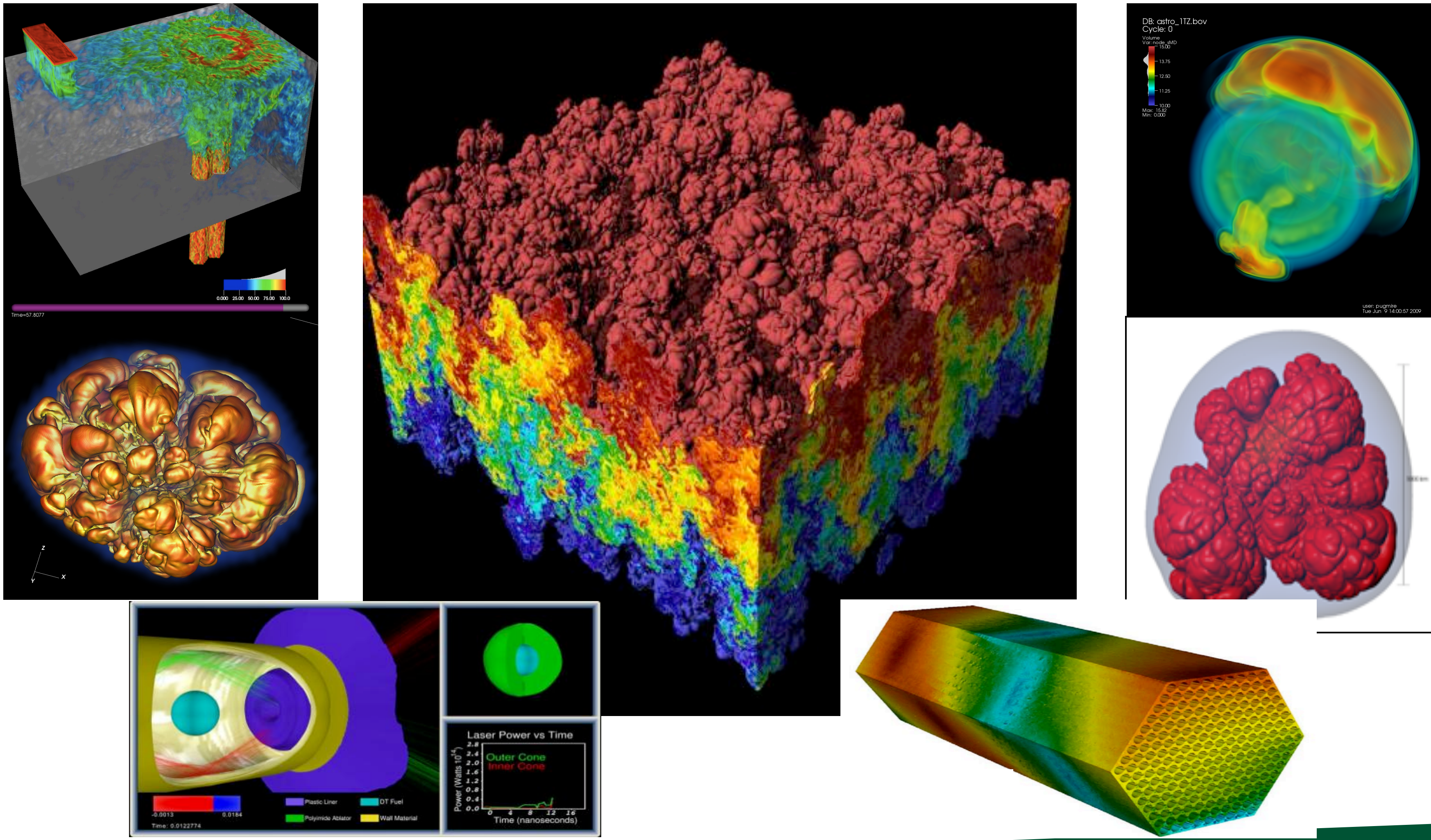
CIS 441/541: Intro to Computer Graphics
Lecture 11: ModelView



May 11, 2021

Hank Childs, University of Oregon

Office Hours

Published | Edit

How to access Office Hours

Hank Childs

Apr 4 at 2:02pm

All Sections

Hi Everyone,

We currently have an asymmetry for accessing Hank and Abhishek's Office Hours.

As of now, Abhishek's are always at: COVERED UP (THIS IS POSTED ONLINE)

And Hank's are accessible via the Zoom Meetings area in Canvas.

Let's chat on Tuesday about the most standard way to do this.

Finally, here is the OH schedule again:

Monday (Abhishek): 10am-11am
Tuesday (Abhishek): 945am-1045am
Wednesday (Hank): 230pm-330pm
Thursday (Abhishek): 945am-1045am

Best,
Hank

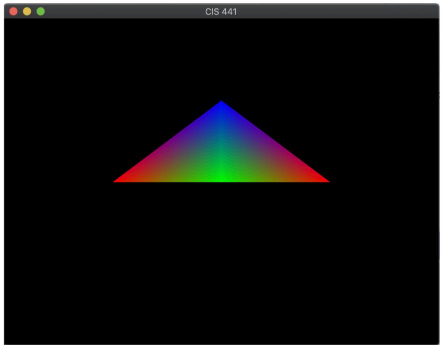
Quiz Thursday

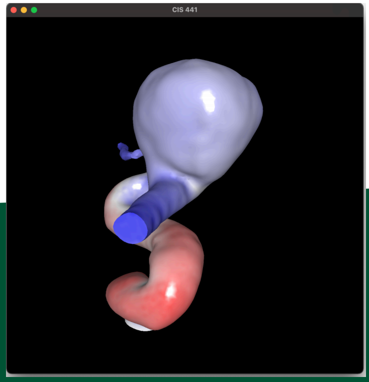
- Phong shading
- Personalized quiz
- Open book, open notes
- Calculator OK

Questions on 2A?

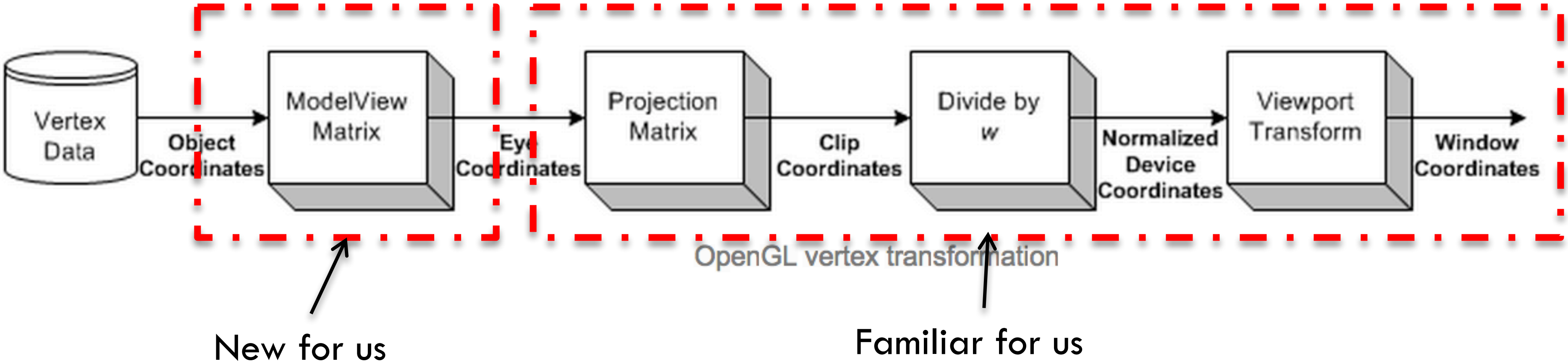
Project 2A

- Assigned Tuesday, due in 5 days (Tuesday May 11)
- Worth 8% of your grade
- Implementing Project 1 within OpenGL
- 5 phases
  - Phase 1: install GLFW
  - Phase 2: run example program
  - Phase 3: modify VBO/VAO
  - Phases 4 & 5: shader programs
- Please start ASAP on Phase 1-3
- Thursday's lecture will be on Phase 4 & 5





ModelView and Projection Matrices



- ModelView idea: two purposes … model and view
  - Model: extra matrix, just for rotating, scaling, and translating geometry.
    - How could this be useful?
  - View: Cartesian to Camera transform

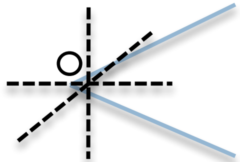
"Model" Part of ModelView

Add additional transforms here….

World space:

Triangles in native Cartesian coordinates

Camera located anywhere



Camera space:

Camera located at origin, looking down -Z

Triangle coordinates relative to camera frame

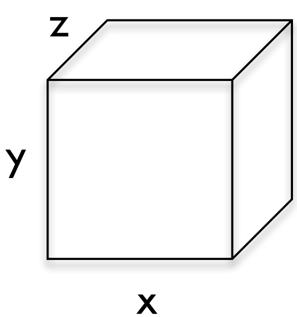
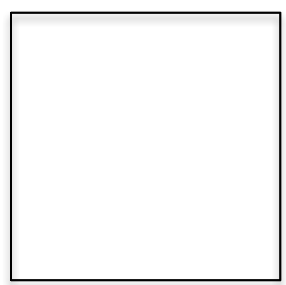


Image space:

All viewable objects within

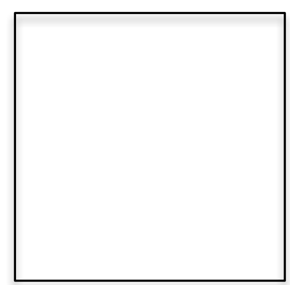
$-1 <= x,y,z <= +1$



Screen space:

All viewable objects within

$-1 <= x, y <= +1$



Device space:

All viewable objects within

$0<=x<=width$, $0 <=y<=height$

How does ModelView work in GL?

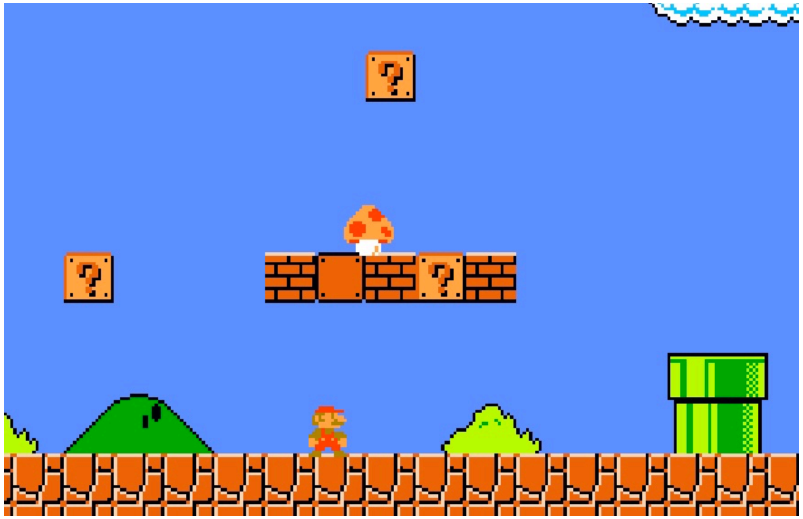
- Determine the matrix
  - Determine model part
  - Determine view part
  - Combine them
- Tell OpenGL about the matrix
- Vertex shader uses the matrix

How does ModelView work in GL?

- Determine the matrix
  - Determine model part
  - Determine view part
  - Combine them
- Tell OpenGL about the matrix
- Vertex shader uses the matrix

Determining the Model Transform

- Typical plan:
  - Have geometric model
    - Came from a file, centered at origin
  - Need to “move” it into position
    - Done with a 4x4 matrix
  - Specifics are the focus of today’s lecture



How does ModelView work in GL?

- Determine the matrix
  - Determine model part
  - Determine view part
  - Combine them
- Tell OpenGL about the matrix
- Vertex shader uses the matrix

Determining the View Transform

- Set up same matrices we did for 1E
- Now we use "glm" – a library for OpenGL matrices

How does ModelView work in GL?

- Determine the matrix
  - Determine model part
  - Determine view part
  - Combine them
- Tell OpenGL about the matrix
- Vertex shader uses the matrix

Combining Model and View

- (4x4 matrix) times (4x4 matrix) → 4x4 matrix

Combining Model and View: Conventions

- For a vertex P
- For a model transformation M
- For a view transformation V
- Two conventions
  - P*M*V = P'
  - P' = V*M*P
- We are using the second convention
  - This is important, more detail later

How does ModelView work in GL?

- Determine the matrix
  - Determine model part
  - Determine view part
  - Combine them
- Tell OpenGL about the matrix
- Vertex shader uses the matrix

Game Plan

- Make a uniform for the ModelView

```
mvploc = glGetUniformLocation(shaderProgram, "MVP");
```

- OpenGL does not know this matrix is "special"
- Set the uniform every time time ModelView changes

```
void RenderManager::MakeModelView(glm::mat4 &model)
{
   glm::mat4 modelview = projection * view * model;
   glUniformMatrix4fv(mvploc, 1, GL_FALSE, &modelview[0][0]);
}
```

- Vertex shader knows to look for the uniform and use it

How does ModelView work in GL?

- Determine the matrix
  - Determine model part
  - Determine view part
  - Combine them
- Tell OpenGL about the matrix
- Vertex shader uses the matrix

Vertex Shader Uses the Matrix

```
const char *GetVertexShader()
{
   static char vertexShader[1024];
   strcpy(vertexShader,
           "#version 400\n"
           "layout (location = 0) in vec3 vertex_position;\n"
           "uniform mat4 MVP;\n"
           "void main() {\n"
           "  gl_Position = MVP*vec4(vertex_position, 1.0);\n"
           "}\n"
         );
   return vertexShader;
}
```

New Topic: Types of Model Transforms

- Three main types
  - Rotate
  - Translate
  - Scale
- Each can be represented as a 4x4 matrix

Convenience routines in 2B
(which use convenience routines from glm)

```
glm::mat4 RotateMatrix(float degrees, float x, float y, float z)
{
   glm::mat4 identity(1.0f);
   glm::mat4 rotation = glm::rotate(identity,
                                    glm::radians(degrees),
                                    glm::vec3(x, y, z));
   return rotation;
}

glm::mat4 ScaleMatrix(double x, double y, double z)
{
   glm::mat4 identity(1.0f);
   glm::vec3 scale(x, y, z);
   return glm::scale(identity, scale);
}

glm::mat4 TranslateMatrix(double x, double y, double z)
{
   glm::mat4 identity(1.0f);
   glm::vec3 translate(x, y, z);
   return glm::translate(identity, translate);
}
```

Combining Model Transforms

- You don't have to choose just 1
- Assume you have a model for a chess rook
  - Possibly need to scale it
  - Almost certainly need to translate it
  - Likely don't need to rotate it

- And: a different transform for each chess piece
- Game plan: use multiple matrices, combine to make one big operation
- But: order matters

Which of two of these three are the same?

- Choice A:
  - Scale(2, 2, 2);
  - Translate(1, 0, 0);
- Choice B:
  - Translate(1, 0, 0);
  - Scale(2, 2, 2);
- Choice C:
  - Translate(2, 0, 0);
  - Scale(2, 2, 2);

SLIDE REPEAT:

Combining Model and View: Conventions

- For a vertex P
- For a model transformation M
- For a view transformation V
- Two conventions
  - P*M*V = P'
  - P' = V*M*P
- We are using the second convention
  - This is important, more detail later

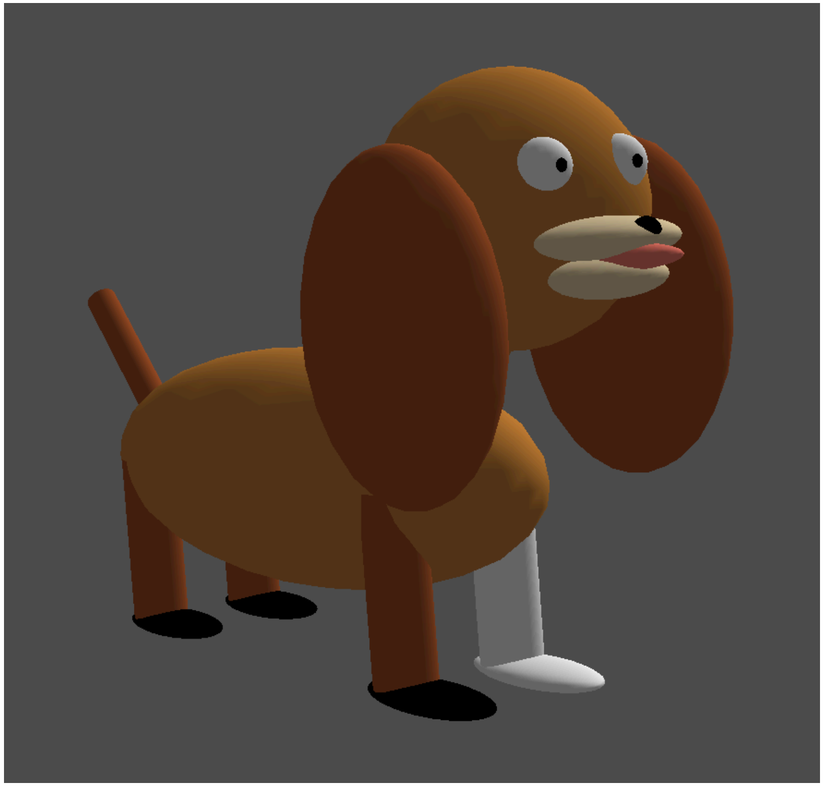
Multiple Model Transforms

- Let M1 be the first transform
- Let M2 be the second transform
- Then the combined model transform should be M2*M1
  - And not M1*M2
- In all:
  - V * M2 * M1 * P → P'
- Make sure you think about order when you do 2B!

Project 2B

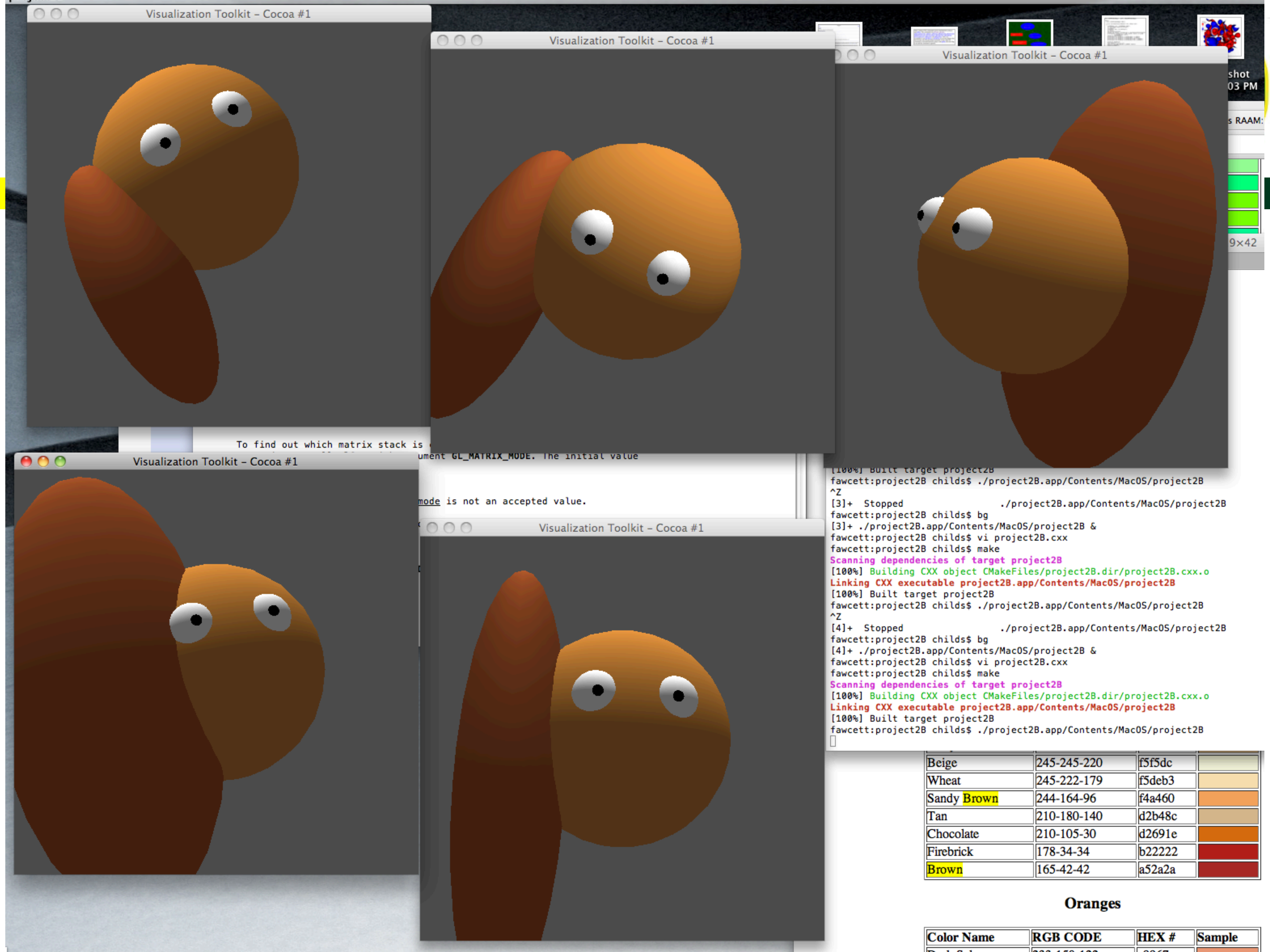
Project #2B (7%), Due Monday May 17th

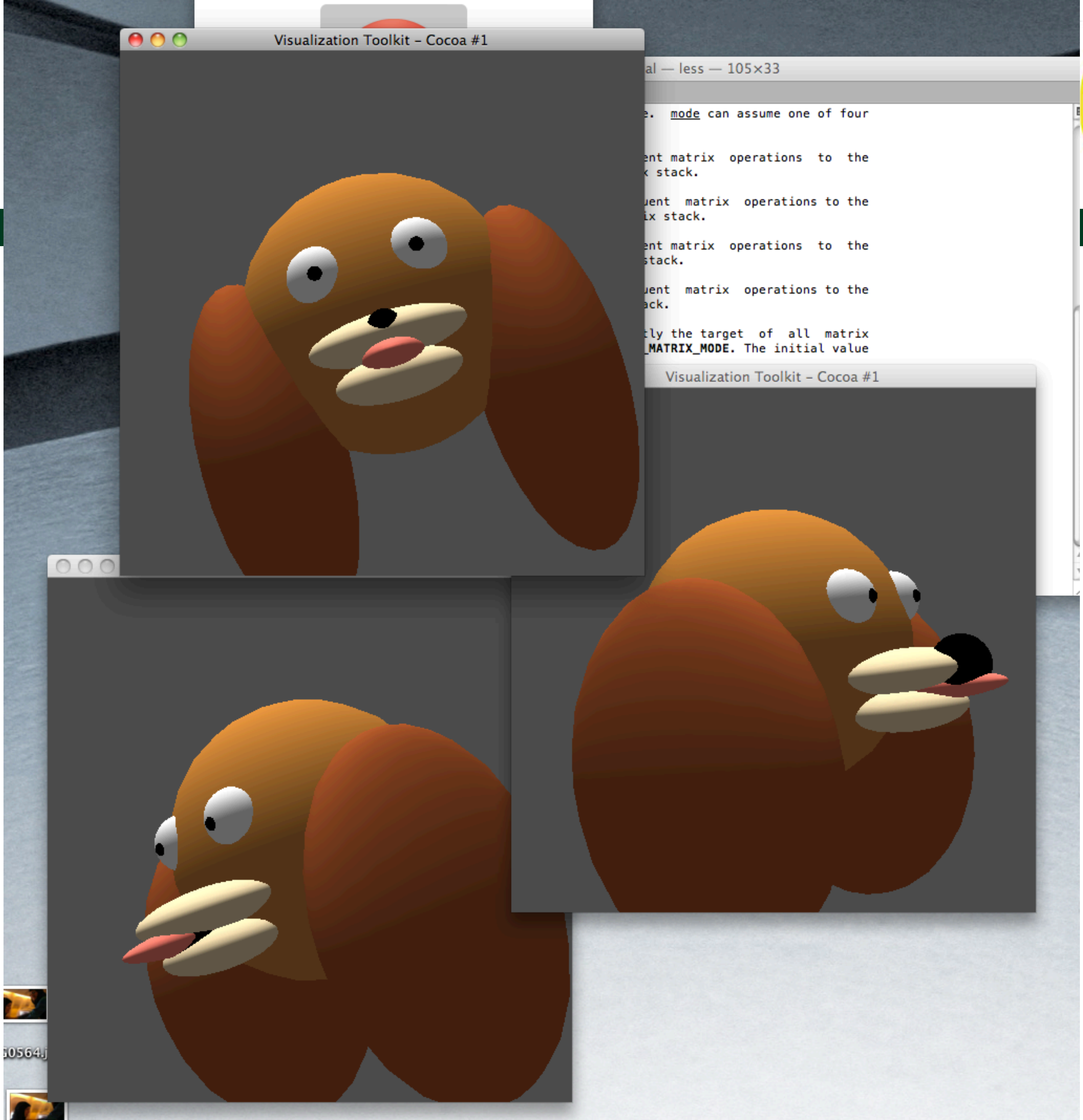
- Goal: modify ModelView matrix to create dog out of spheres and cylinders
- New code skeleton: "project2B.cxx"
- No geometry file needed
- You will be able to do this by rendering ~20 spheres and cylinders, each with their own transform

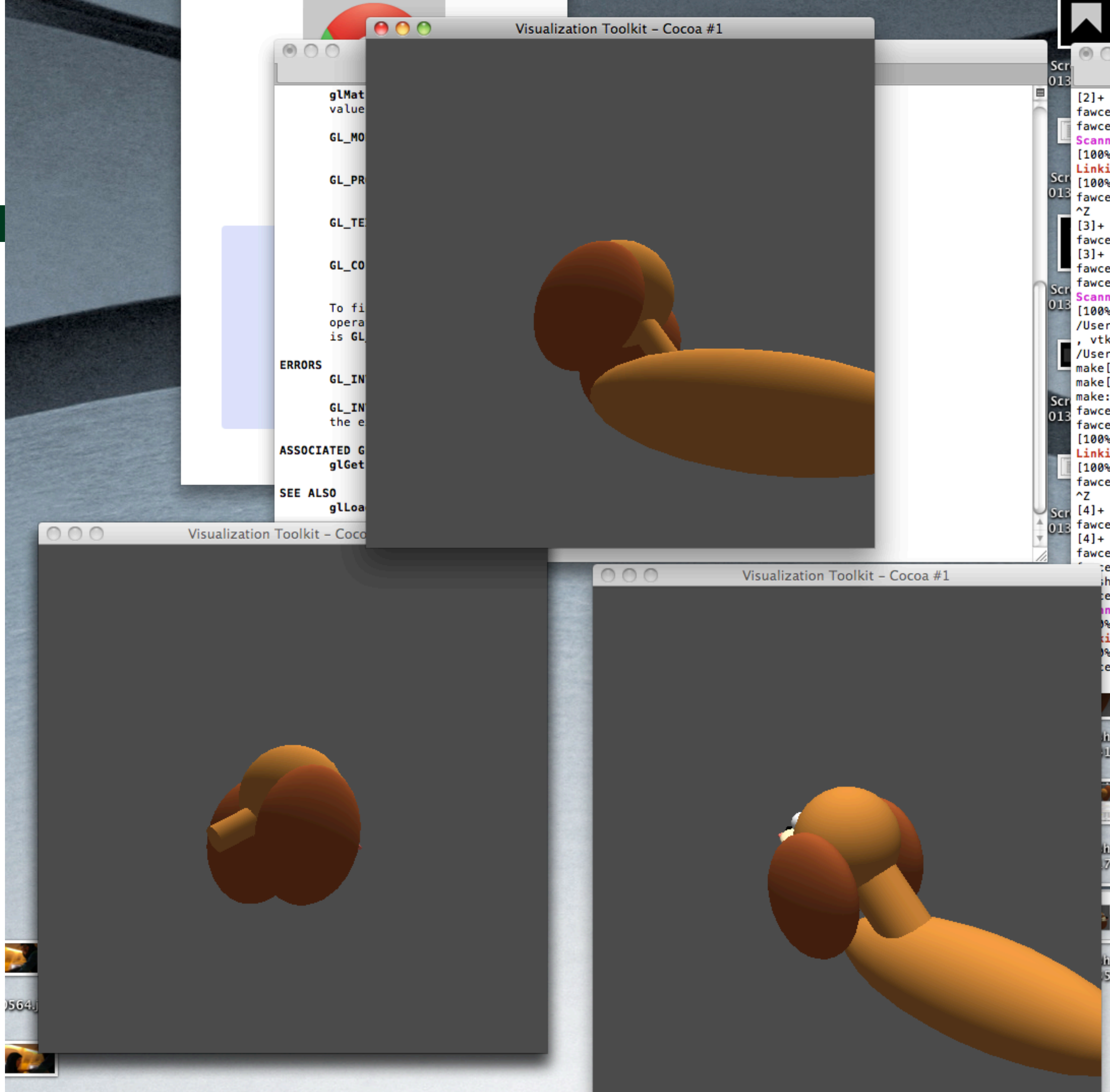


What is the correct answer?

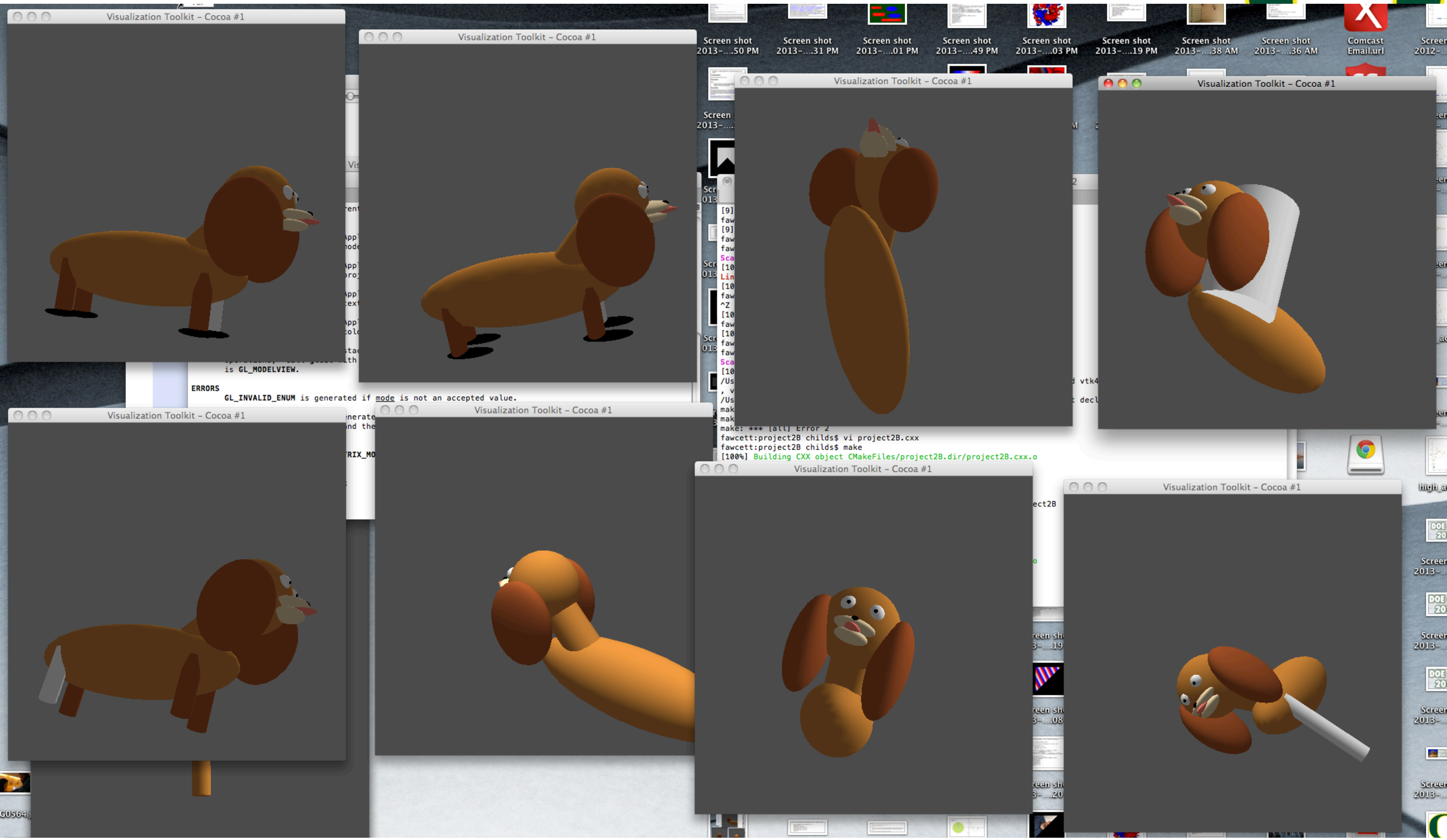
- The correct answer is:
  - Something that looks like a dog
    - No obvious problems with output geometry
  - Something that uses the sphere and cylinder classes
    - If you use something else, please clear it with me first
      - I may reject your submission if I think you are using outside resources that make the project too easy
  - Something that uses rotation
    - For me: the neck and tail
  - Something that animates
- Aside from that, feel free to be as creative as you want … color, breed, etc.


Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1
Visualization Toolkit – Cocoa #1
To find out which matrix stack is
GL_MATRIX_MODE. The initial value
mode is not an accepted value.
fawcett:project2B childs$ ./project2B.app/Contents/MacOS/project2B
^Z
[3]+ Stopped ./project2B.app/Contents/MacOS/project2B
fawcett:project2B childs$ bg
[3]+ ./project2B.app/Contents/MacOS/project2B &
fawcett:project2B childs$ vi project2B.cxx
fawcett:project2B childs$ make
Scanning dependencies of target project2B
[100%] Building CXX object CMakeFiles/project2B.dir/project2B.cxx.o
Linking CXX executable project2B.app/Contents/MacOS/project2B
[100%] Built target project2B
fawcett:project2B childs$ ./project2B.app/Contents/MacOS/project2B
^Z
[4]+ Stopped ./project2B.app/Contents/MacOS/project2B
fawcett:project2B childs$ bg
[4]+ ./project2B.app/Contents/MacOS/project2B &
fawcett:project2B childs$ vi project2B.cxx
fawcett:project2B childs$ make
Scanning dependencies of target project2B
[100%] Building CXX object CMakeFiles/project2B.dir/project2B.cxx.o
Linking CXX executable project2B.app/Contents/MacOS/project2B
[100%] Built target project2B
fawcett:project2B childs$ ./project2B.app/Contents/MacOS/project2B
Beige 245-245-220 f5f5dc
Wheat 245-222-179 f5deb3
Sandy Brown 244-164-96 f4a460
Tan 210-180-140 d2b48c
Chocolate 210-105-30 d2691e
Firebrick 178-34-34 b22222
Brown 165-42-42 a52a2a
Oranges
Color Name RGB CODE HEX # Sample

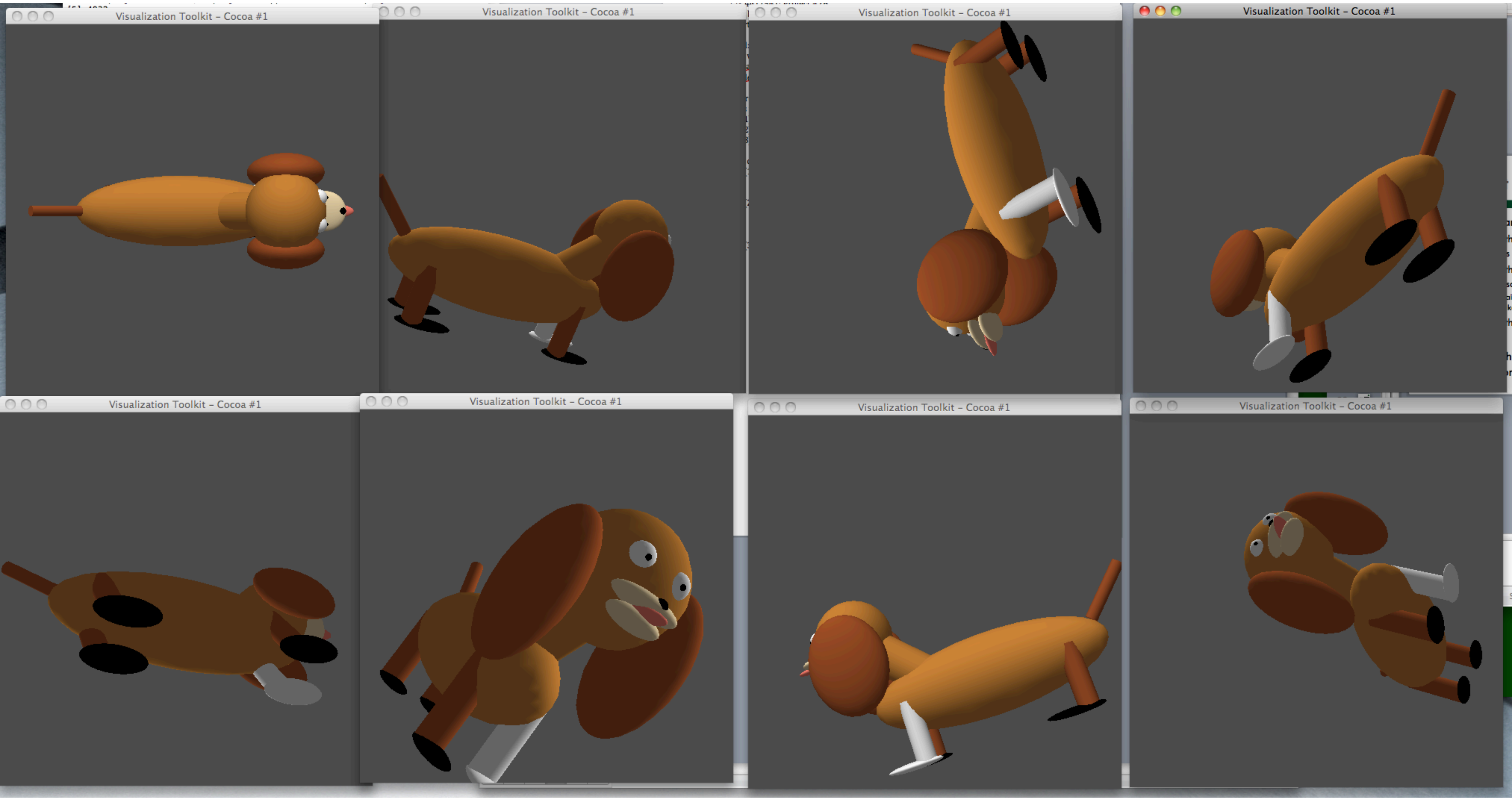

Visualization Toolkit – Cocoa #1
al — less — 105×33
e. mode can assume one of four
ent matrix operations to the
x stack.
uent matrix operations to the
ix stack.
ent matrix operations to the
stack.
uent matrix operations to the
ack.
tly the target of all matrix
_MATRIX_MODE. The initial value
Visualization Toolkit – Cocoa #1
50564.j



Visualization Toolkit – Cocoa #1
Visualization Toolkit – Coco
Visualization Toolkit – Cocoa #1
ERRORS
ASSOCIATED G
SEE ALSO

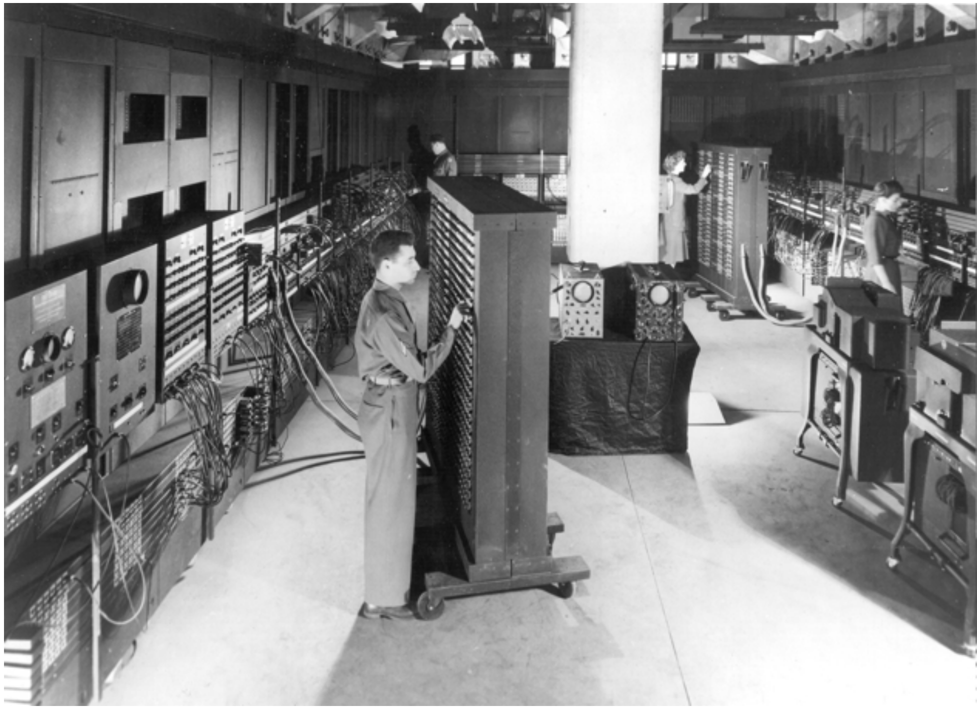

Visualization Toolkit - Cocoa #1
GL_INVALID_ENUM is generated if mode is not an accepted value.
fawcett:project2B childs$ vi project2B.cxx
fawcett:project2B childs$ make
[100%] Building CXX object CMakeFiles/project2B.dir/project2B.cxx.o

For your reference: my dog



New Topic: the Amazing GPU

“First” computer: ENIAC



- Year: 1946
- Location: Pennsylvania
- Purpose: military
- Cost: $487K
  - ($6.9M today)
- Technology:
  - very different than today
  - … but still the same

Vacuum Tubes

- Vacuum tubes:
  - Glass tubes with no gas
  - Used to control electron flow in early computers
- Occasionally, a bug would get stuck in the tube and cause the program to malfunction
- We no longer have vacuum tubes, but the term bug has remained with us…

Vacuum tubes in ENIAC
Image source: wikipedia

An ENIAC Computation

- Used for military calculations:
  - A-bomb design
  - Missile delivery
- ENIAC could do ~5000 calculations in one minute
- In one case:
  - ENIAC did a calculation in 30 seconds
  - Human being took 20 hours
  - 2400x increase in speed

source: wikipedia

Hertz (Hz) = unit of measurement for how fast you do something

- 1 Hertz = do something once per second
- KHz = 1024 Hz
- MHz = 1024 KHz
- GHz = 1024 MHz

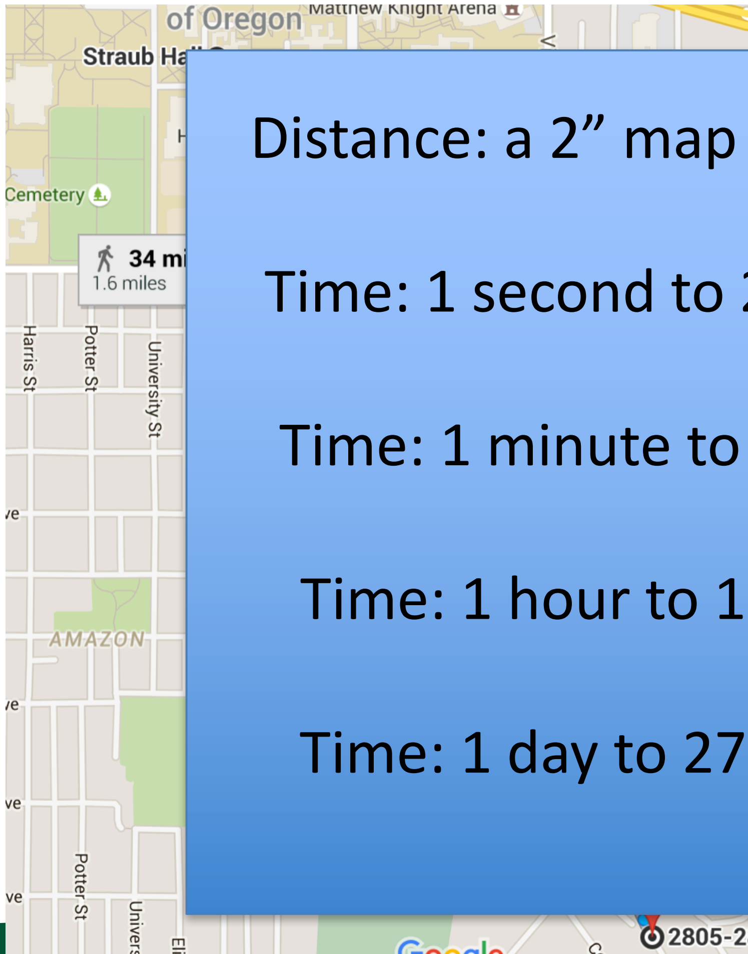
- The ENIAC machine ran at 5000Hertz, or about 5KHz.
  - Vocab term: “clock speed” → the number of cycles per second
    - the clock speed of the ENIAC was 5KHz

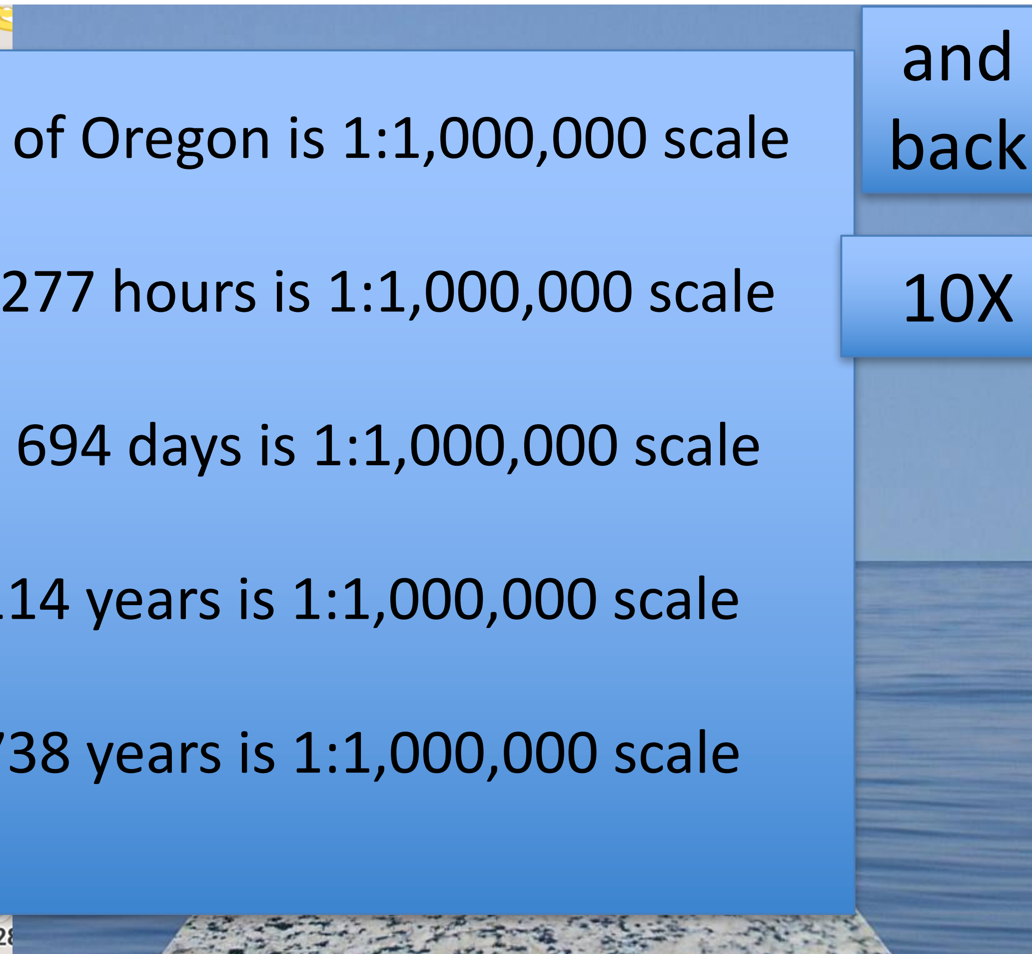
Today's Desktop Computers Are Fast!

- Most computers run at ~1-3 GHz
- i.e., operates billions of instructions each second
- This is about one million times faster than the ENIAC
  - … and the ENIAC was 2400X faster than humans
  - (at least at tasks computers are good at)

UNIVERSITY OF OREGON

What does a million-fold increase mean?





Distance: a 2” map of Oregon is 1:1,000,000 scale

Time: 1 second to 277 hours is 1:1,000,000 scale

Time: 1 minute to 694 days is 1:1,000,000 scale

Time: 1 hour to 114 years is 1:1,000,000 scale

Time: 1 day to 2738 years is 1:1,000,000 scale

and back

10X

1 million-fold increase! How does this happen?

- Moore’s Law (old timer’s version)
  - Clock speed doubles every 18 months
- Moore’s Law (newer version but still for old timers)
  - Clock speed doubles every 24 months

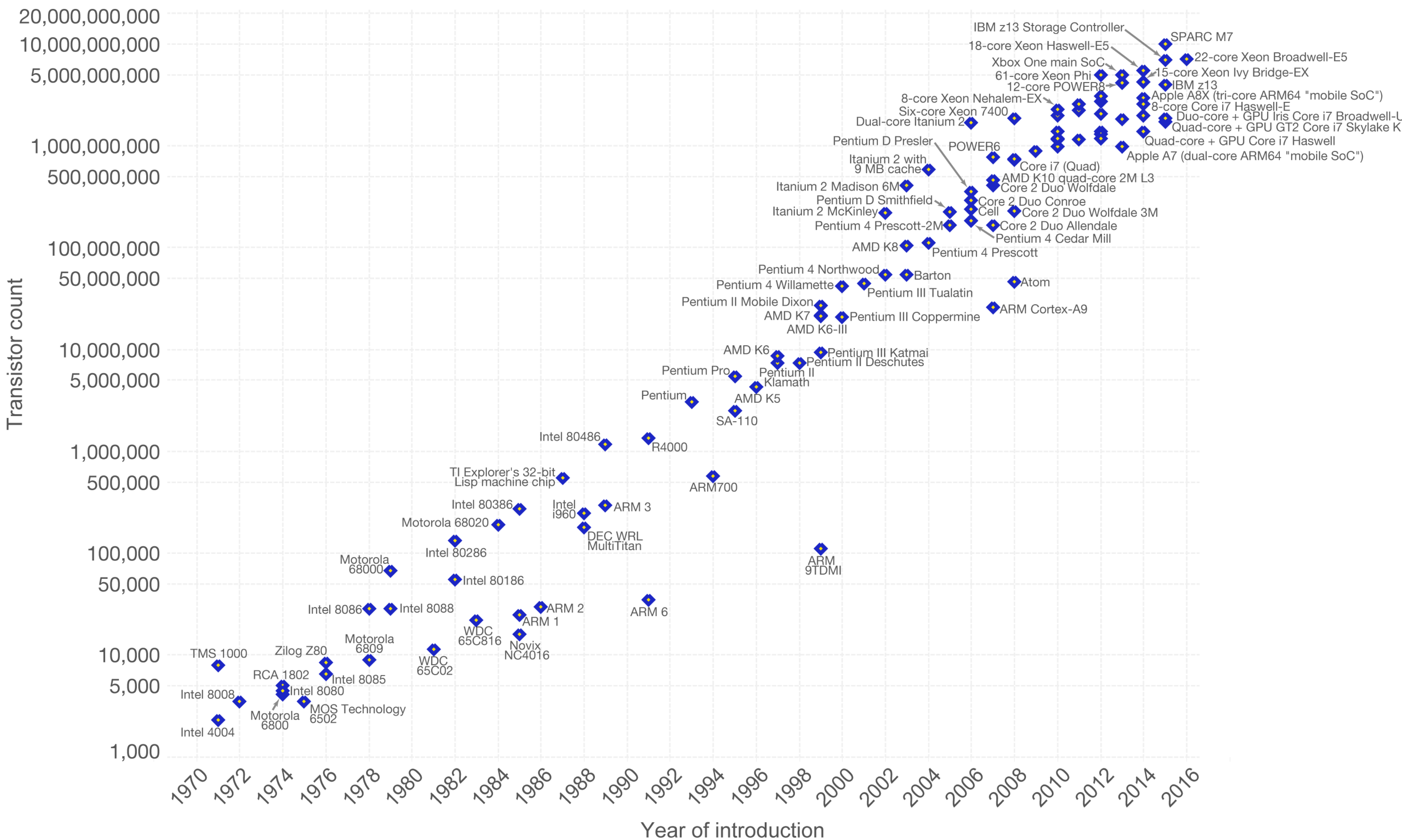
Moore's Law

- Moore's Law (actual version)
  - Number of transistors doubles every 24 months
  - And clock speed is a reflection of number of transistors
- So what is a transistor?
  - Semiconductor device for amplifying or switching electronic signals/power
  - Fundamental building block of modern electronics
  - Replacement for vacuum tube

Moore's Law – The number of transistors on integrated circuit chips (1971-2016)

Our World in Data

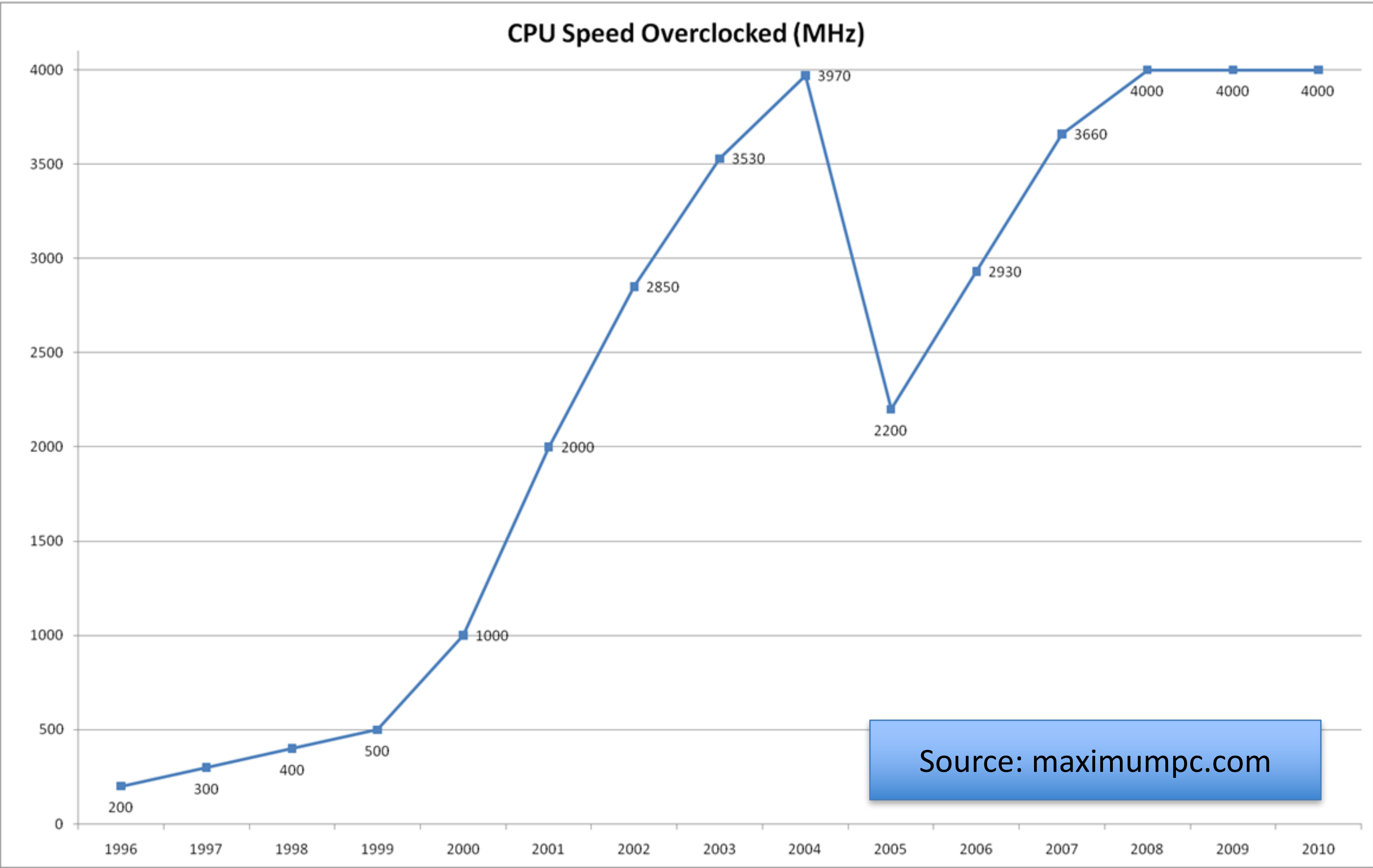
Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)

The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

But actually…

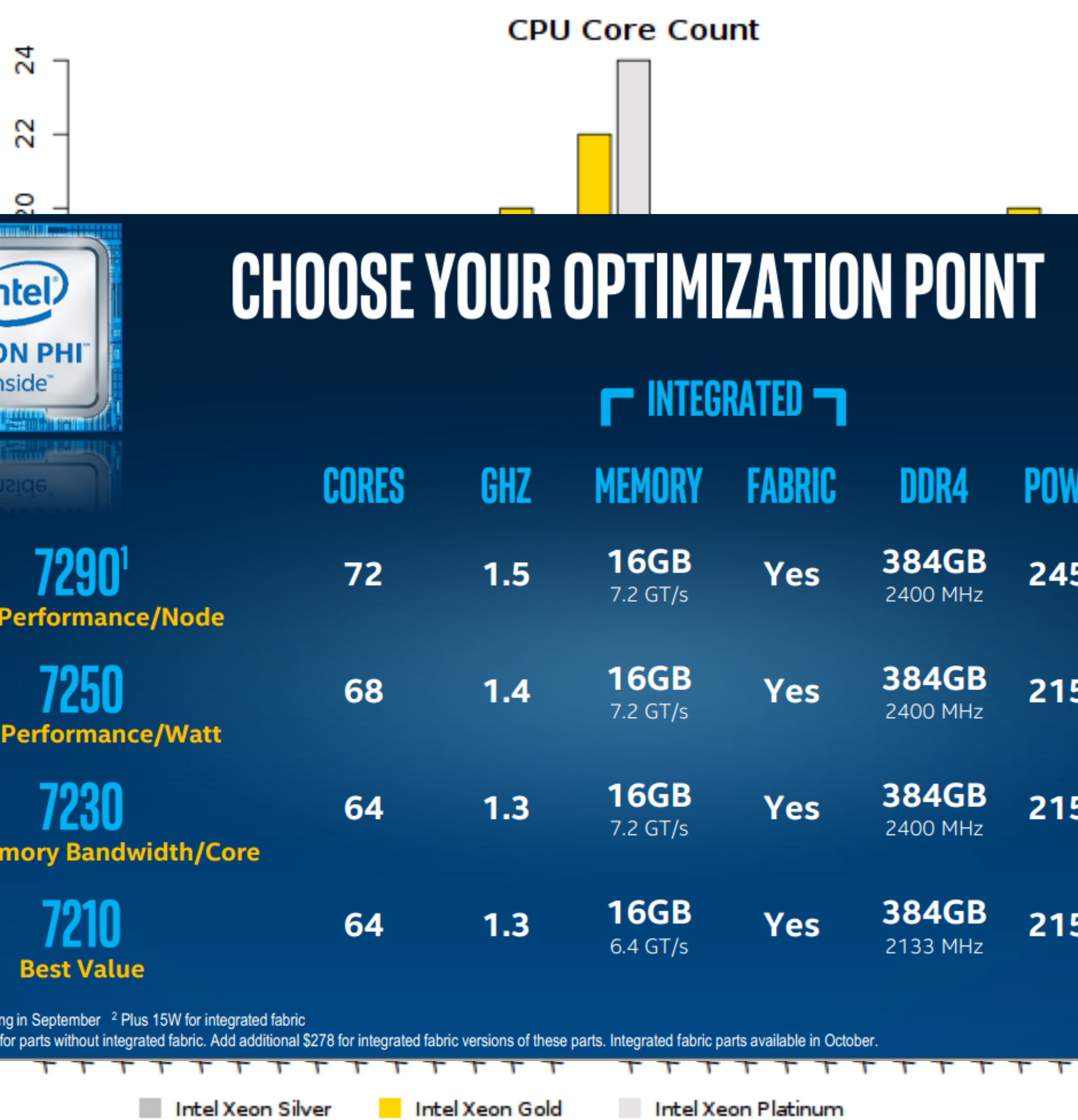


The reason is power

- Desktop computer takes ~200W
  - There are multiple components that consume the power:
    - CPU
    - Monitor
    - Disk
    - Memory
- 200W * 1 year → ~$70

Relationship Between Power and Clock Speed

- Clock goes twice as fast → Power goes up by factor of 8
  - (Increase of X in clock speed → Increase of X^3 in power)
- Clock speeds haven't changed in 12 years
- What if they had doubled every 2 years?
- Then 64X faster
  - → 262144X more power (for the CPU)
  - → power bill now $18M



CHOOSE YOUR OPTIMIZATION POINT

intel XEON PHI inside

	Cores	GHz	Integrated Memory	Integrated Fabric	DDR4	Power[2]	Recommended Customer Pricing[3]
7290[1] Best Performance/Node	72	1.5	16GB 7.2 GT/s	Yes	384GB 2400 MHz	245W	$6254
7250 Best Performance/Watt	68	1.4	16GB 7.2 GT/s	Yes	384GB 2400 MHz	215W	$4876
7230 Best Memory Bandwidth/Core	64	1.3	16GB 7.2 GT/s	Yes	384GB 2400 MHz	215W	$3710
7210 Best Value	64	1.3	16GB 6.4 GT/s	Yes	384GB 2133 MHz	215W	$2438

[1] Available beginning in September [2] Plus 15W for integrated fabric
[3] Pricing shown is for parts without integrated fabric. Add additional $278 for integrated fabric versions of these parts. Integrated fabric parts available in October.

M - an optional SKU is available with support for up to 1.5TB memory per CPU socket

F - an optional SKU is available with integrated 100Gbps Intel Omni-Path fabric

How To Use Multiple Cores?

- Answer: parallel programming
  - Write computer programs that use all the cores
  - Ideally the coordination between the cores is minimal

Parallel Programming Concepts

- Usual goal:
  - if program takes N seconds to run with one core
  - then take N/2 seconds to run with two cores
  - and N/M seconds to run with M cores

Let's consider an example outside of computers

Example: paint a house

- One person: 6 days (1 day = 10 hours)
- Two people: 3 days
- Three people: 2 days
- Six people: 1 day

- Sixty people: 1 hour?
- Six hundred people: 6 minutes?

Example: paint a house, plan #2

Parallel programming is hard, and smart people spend their whole careers figuring out how to make parallel programs be efficient

GPUs: Graphical Processing Units (graph

- Historical:
  - Introduced to accelera
  - Boom with desktop PC
  - Mid-2000's: people sta program a GPU to mak
  - Late 2000's: GPU make encouraging folks to p
    - GPGPU: General-purpo
  - Mid 2010's: GPUs used problems.
    - Machine learning work

Emergent Tech

Bitcoin heist with a twist: This time it's servers that were stolen

Icelandic cops cuff 11 on suspicion of data centre robberies

By Simon Sharwood, APAC Editor 5 Mar 2018 at 04:57 19 SHARE ▼

Icelandic police have cuffed 11 people in connection with four raids on data centres that targeted cryptocurrency mining equipment.

The raids started in December 2017 when three data centres were cracked in December. Another raid took place in January. 600 servers went missing in the heists.

Icelandic police kept the raids secret while they pursued their investigations. Those efforts culminated in 11 arrests and an appearance before the Reykjanes District Court last Friday. Two of the 11 were detained and the matter held over for another day.

The 600 servers are all still missing, the Associated Press reports. Which is no surprise: x86 kit is pretty generic. The real prize inside a bitcoin-mining rig is either GPUs, RAM or nicely fast solid-state disks. Those components are all tiny compared to servers and could probably have been posted out of Iceland piecemeal without much hassle.

Iceland has become something of a hub for demanding workloads like cryptocurrency mining because cheap energy and low ambient temperatures make it a low-cost destination to run data centres and the kit inside them. The nation also has a low crime rate. ®

Why Are GPUs So Good?

Market Summary > NVIDIA Corporation

570.63 USD

−21.86 (3.69%) ↓

Closed: May 10, 7:59 PM EDT ·Disclaimer

After hours 567.80 −2.83 (0.50%)

NASDAQ: NVDA

+ Follow

1 day | 5 days | 1 month | 6 months | YTD | 1 year | 5 years | Max

Open	591.49	Mkt cap	355.15B	Prev close	592.49
High	592.24	P/E ratio	82.72	52-wk high	648.57
Low	570.00	Div yield	0.11%	52-wk low	303.79

Graphics and GPUs

- Graphics are very parallelizable
  - How many people can paint a house? <100
  - How many cores can paint a screen? >5000
- GPUs have special support for graphics
  - (Of course they do! ... Graphics processing units!)
- GPUs also have support for general programming
  - Example: Nvidia CUDA

Introduction to Ray Tracing

Dr. Xiaoyu Zhang
Cal State U., San Marcos

Classifying Rendering Algorithms

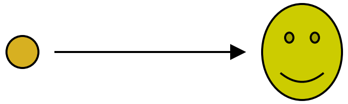
- One way to classify rendering algorithms is according to the type of light interactions they capture
- For example: The OpenGL lighting model captures:
  - Direct light to surface to eye light transport
  - Diffuse and rough specular surface reflectance
  - It actually doesn't do light to surface transport correctly, because it doesn't do shadows
- We would like a way of classifying interactions: *light paths*

Classifying Light Paths

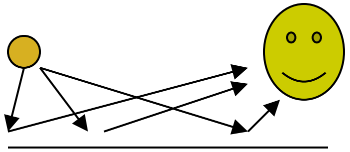
- Classify light paths according to where they come from, where they go to, and what they do along the way
- Assume only two types of surface interactions:
  - Pure diffuse, D
  - Pure specular, S
- Assume all paths of interest:
  - Start at a light source, L
  - End at the eye, E
- Use regular expressions on the letters D, S, L and E to describe light paths
  - Valid paths are L(D|S)*E

Simple Light Path Examples

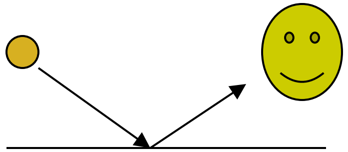
- LE
  - The light goes straight from the source to the viewer



- LDE
  - The light goes from the light to a diffuse surface that the viewer can see

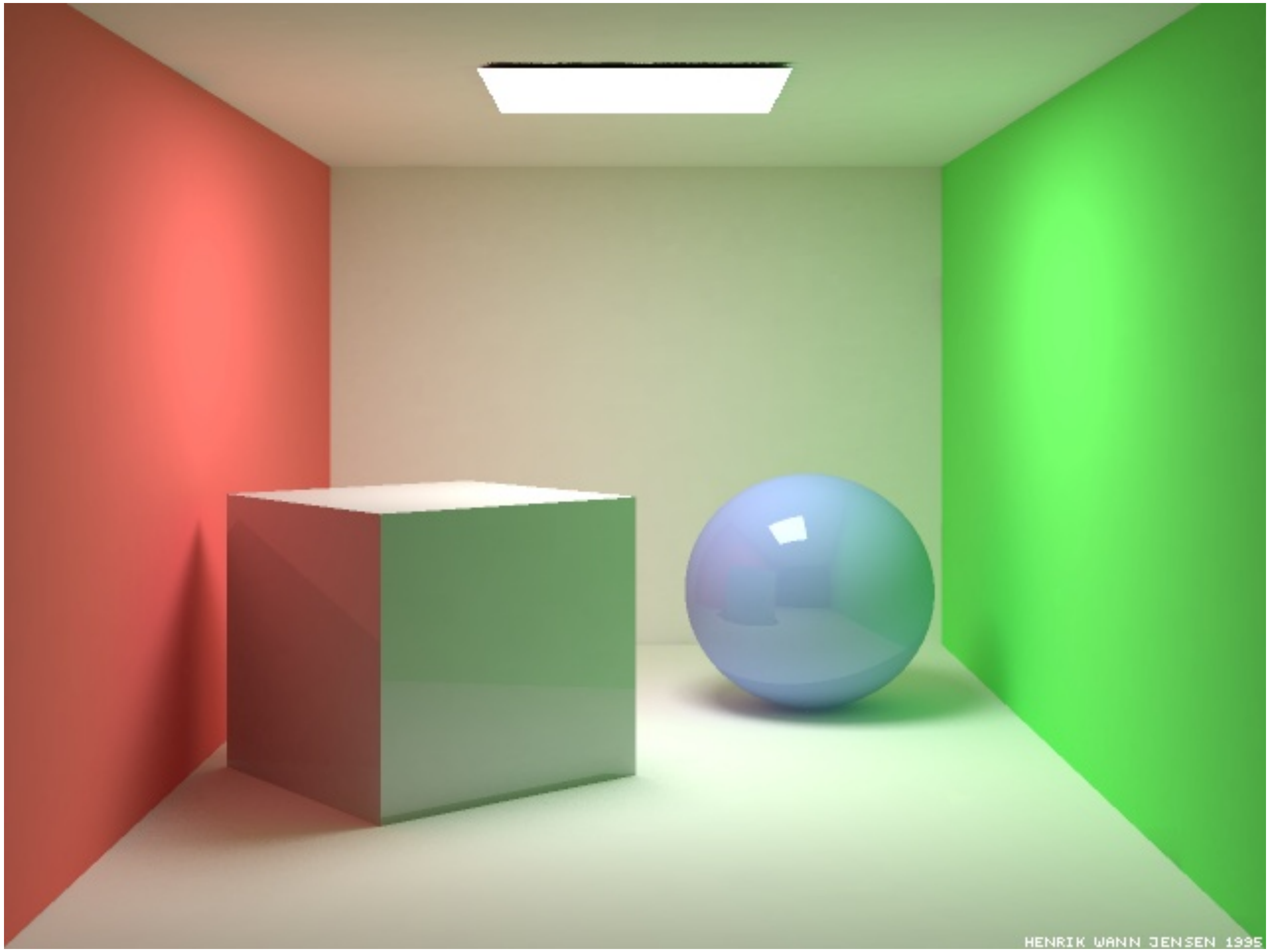


- LSE
  - The light is reflected off a mirror into the viewer’s eyes



- L(S|D)E
  - The light is reflected off either a diffuse surface or a specular surface toward the viewer
- Which do OpenGL (approximately) support?

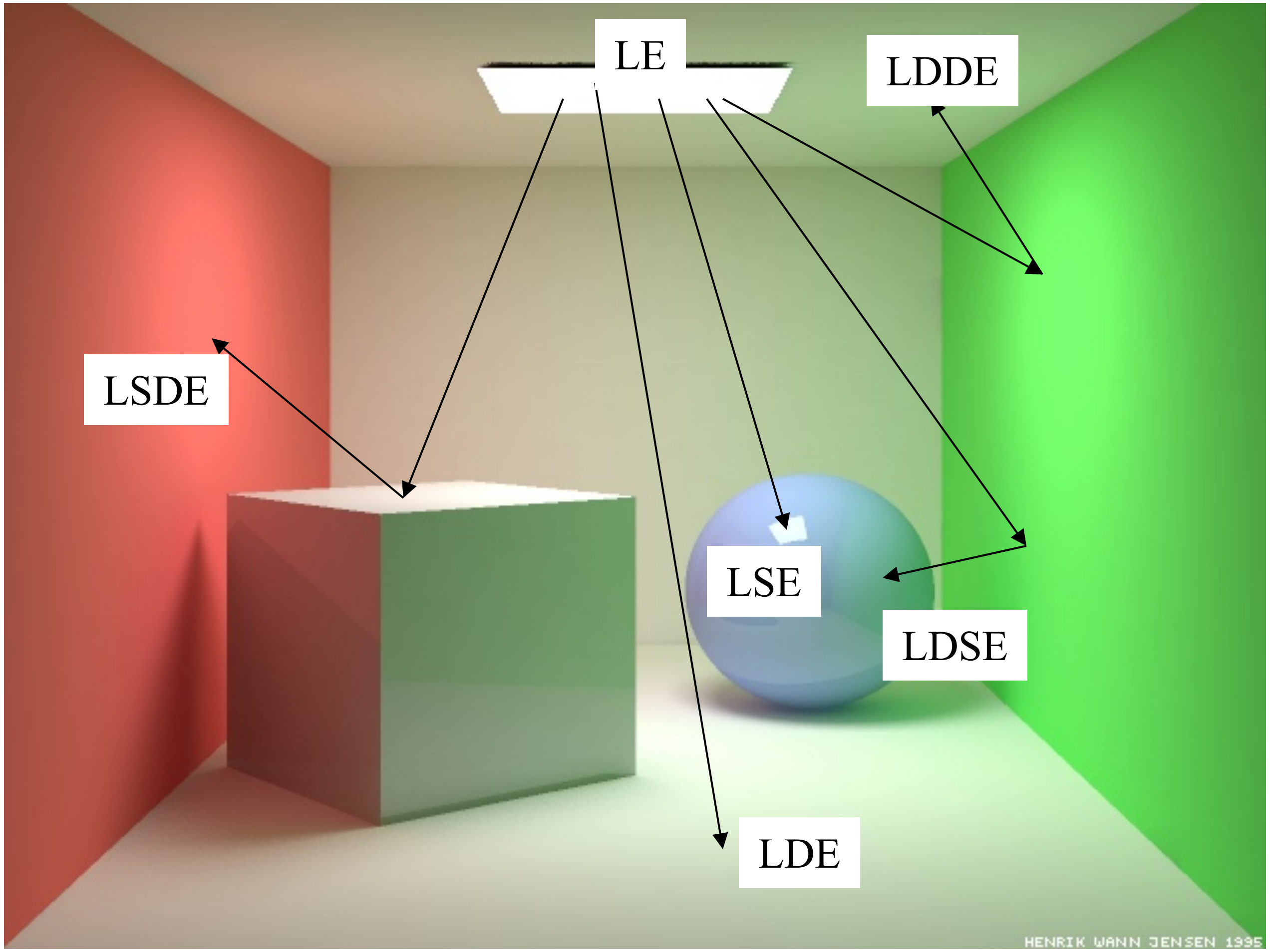
More Complex Light Paths



- Find the following:
  - LE
  - LDE
  - LSE
  - LDDE
  - LDSE
  - LSDE

Radiosity Cornell box, due to Henrik wann Jensen, http://www.gk.dtu.dk/~hwj, rendered with ray tracer

More Complex Light Paths



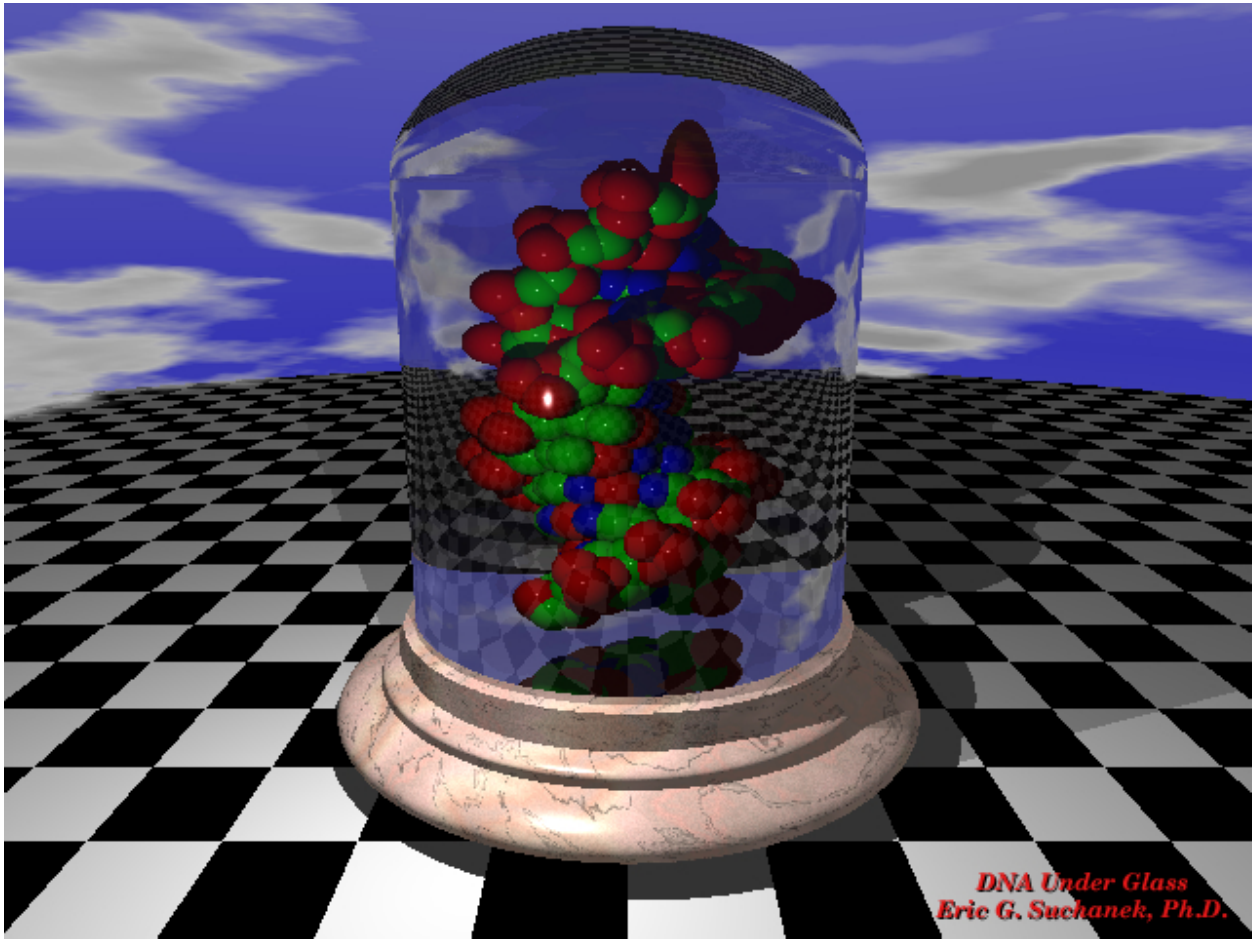
The OpenGL Model

- The “standard” graphics lighting model captures only L(D|S)E
- It is missing:
  - Light taking more than one diffuse bounce: LD*E
    - Should produce an effect called color bleeding, among other things
    - Approximated, grossly, by ambient light
  - Light refracted through curved glass
    - Consider the refraction as a “mirror” bounce: LDSE
  - Light bouncing off a mirror to illuminate a diffuse surface: LS+D+E
  - Many others
  - Not sufficient for photo-realistic rendering

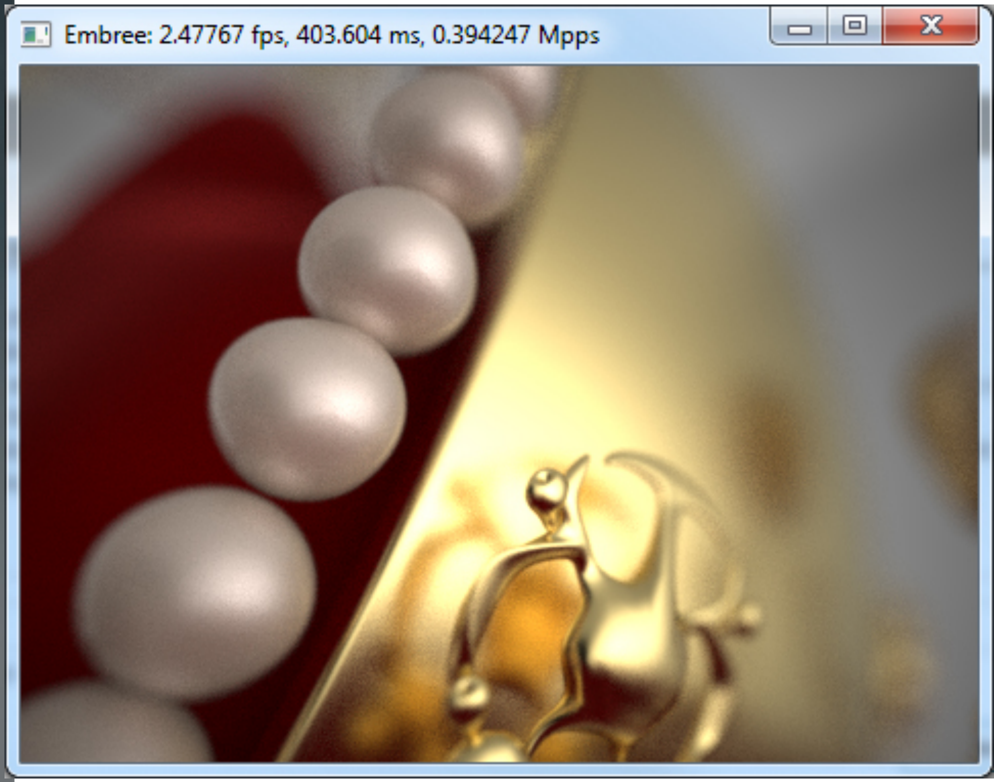
Raytraced Images

PCKTWTCH by Kevin Odhner, POV-Ray

Kettle, Mike Miller, POV-Ray

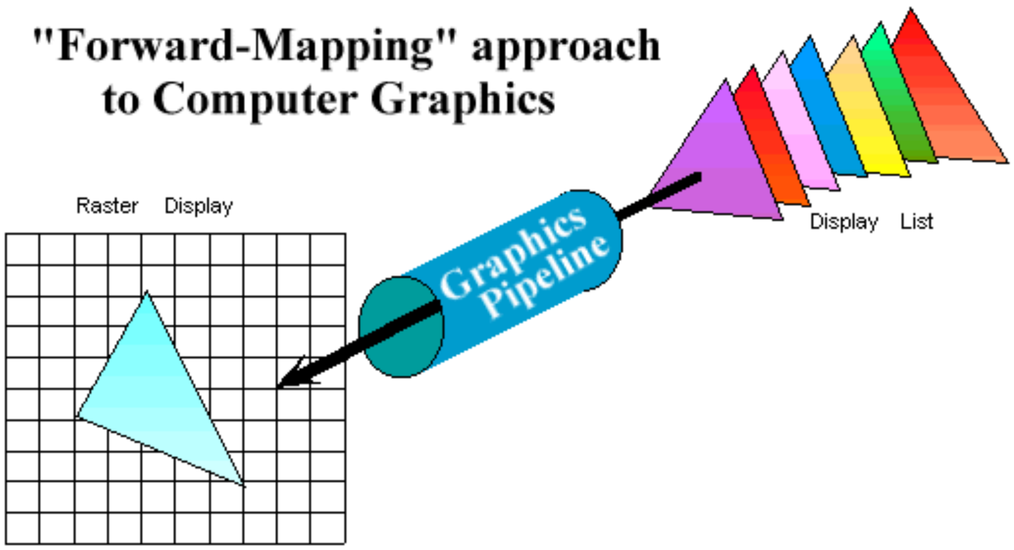

DNA Under Glass
Eric G. Suchanek, Ph.D.

The previous slides now look like amateur hour…



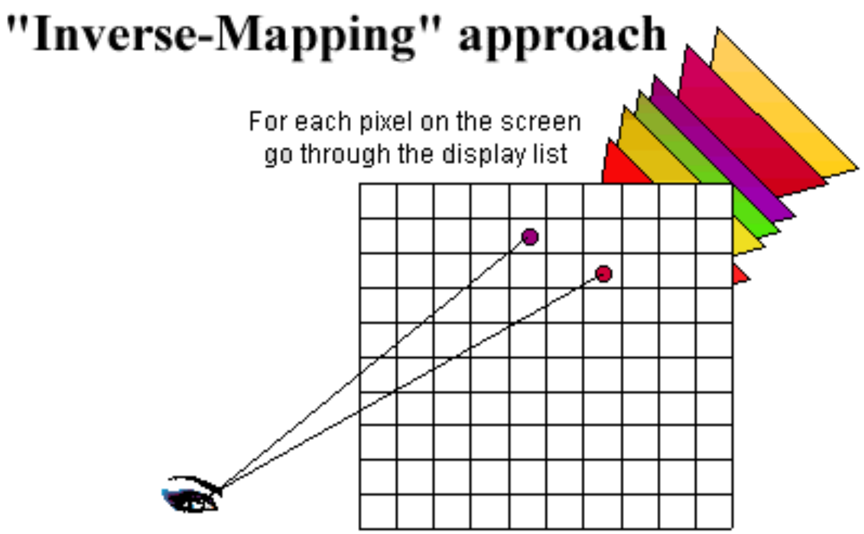
Graphics Pipeline Review

- Properties of the Graphics Pipeline
  - Primitives are transformed and projected (not depending on display resolution)
  - Primitives are processed one at a time
  - Forward-mapping from geometrical space to image space



Alternative Approaches: Ray CASTING (not Ray TRACING)

Ray-casting searches along lines of sight, or rays, to determine the primitive that is visible along it.

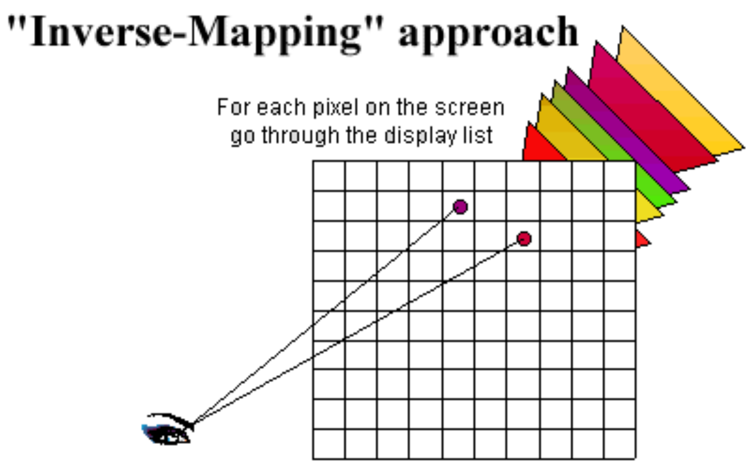


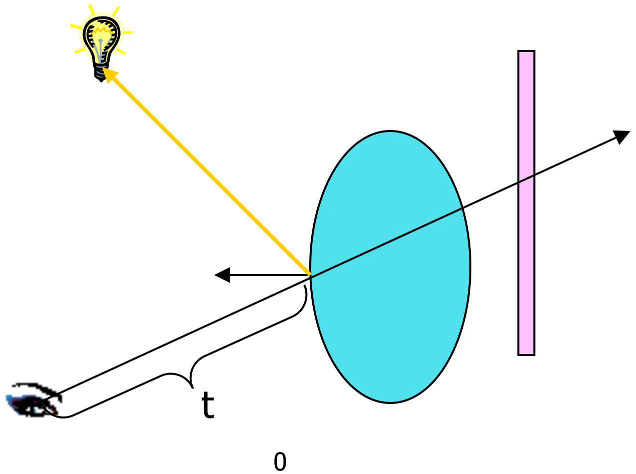
Properties of ray-casting:

- Go through all primitives at each pixel
- Image space sample first
- Analytic processing afterwards

Ray Casting Overview

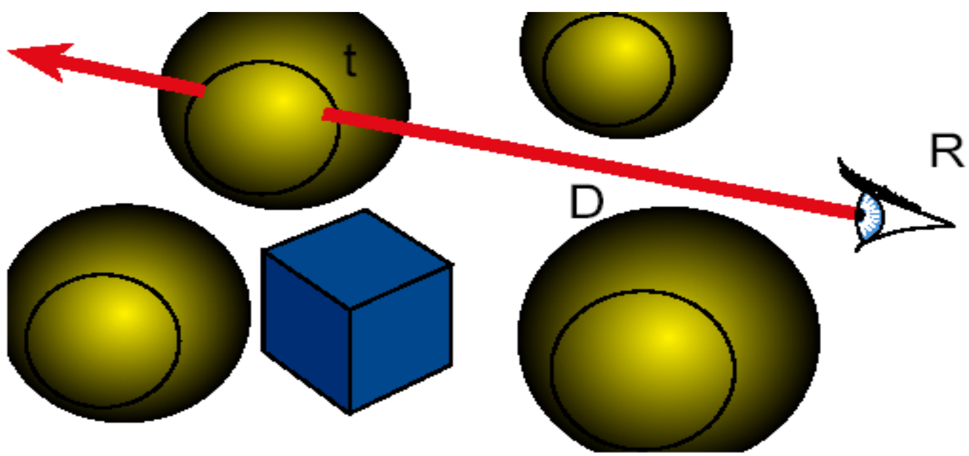
- For every pixel shoot a ray from the eye through the pixel.
- For every object in the scene
  - Find the point of intersection with the ray closest to (and in front of) the eye
  - Compute normal at point of intersection
- Compute color for pixel based on point and normal at intersection closest to the eye (e.g. by Phong illumination model).





Ray Casting

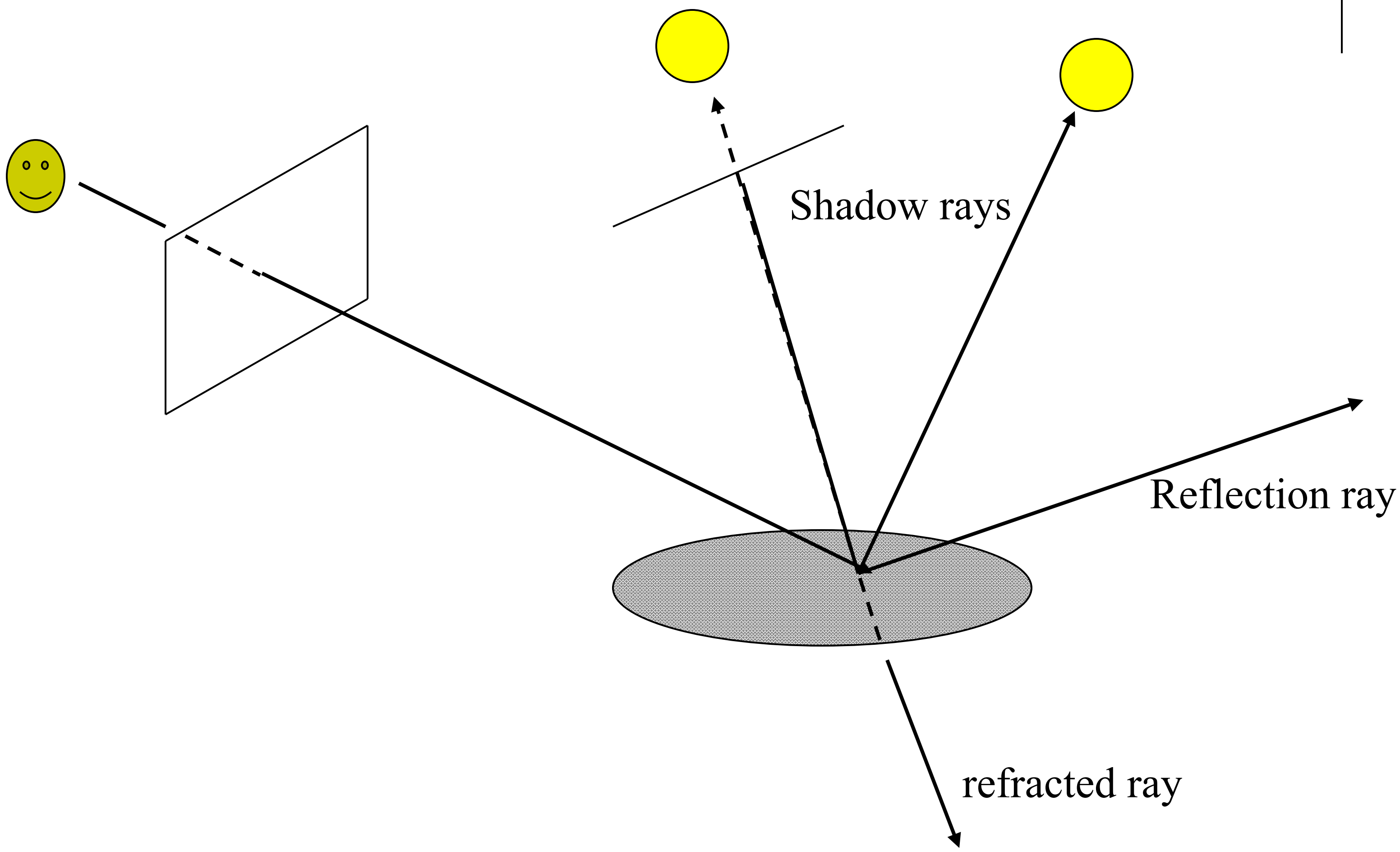
- ```
  Ray Cast ( Point R, Ray D ) {
      foreach object in the scene
        find minimum t>0 such that R + t D hits object
      if ( object hit )
      return object
      else return background object
  }
  ```


  ```

Raytracing

- Cast rays from the eye point the same way as ray casting
  - Builds the image pixel by pixel, one at a time
- Cast additional rays from the hit point to determine the pixel color
  - Shoot rays toward each light. If they hit something, then the object is shadowed from that light, otherwise use "standard" model for the light
  - Reflection rays for mirror surfaces, to see what should be reflected in the mirror
  - Refraction rays to see what can be seen through transparent objects
  - Sum all the contributions to get the pixel color

Raytracing



Recursive Ray Tracing

- When a reflected or refracted ray hits a surface, repeat the whole process from that point
  - Send out more shadow rays
  - Send out new reflected ray (if required)
  - Send out a new refracted ray (if required)
  - Generally, reduce the weight of each additional ray when computing the contributions to surface color
  - Stop when the contribution from a ray is too small to notice or maximum recursion level has been reached

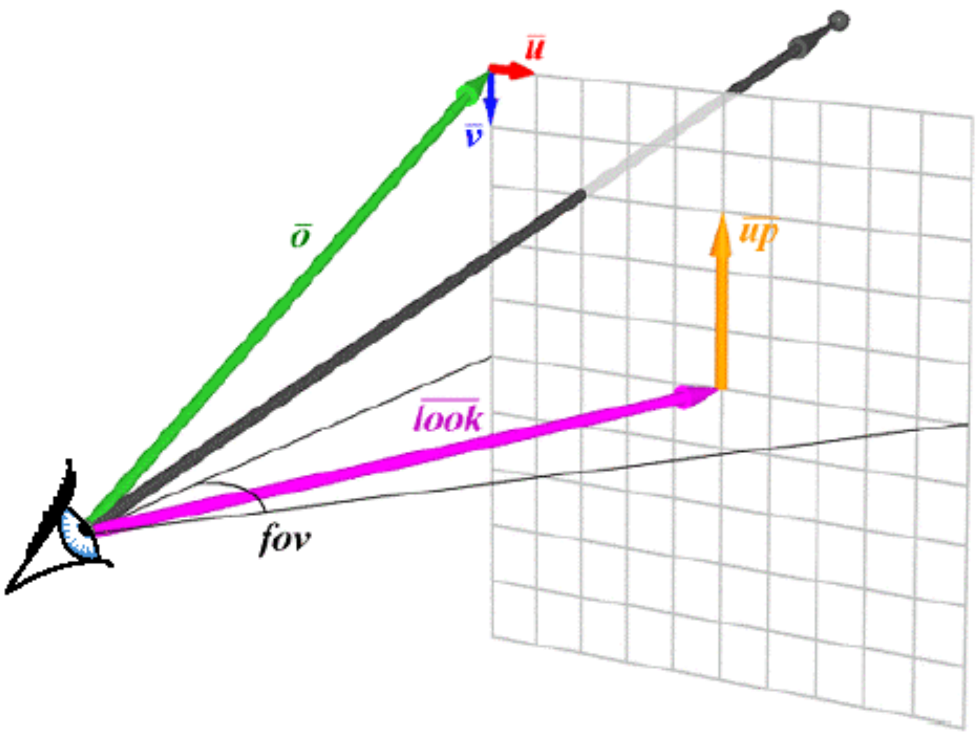
Raytracing Implementation

- Raytracing breaks down into two tasks:
  - Constructing the rays to cast
  - Intersecting rays with geometry
- The former problem is simple vector arithmetic
- Intersection is essentially root finding (as we will see)
  - Any root finding technique can be applied
- Intersection calculation can be done in world coordinates or model coordinates

Constructing Rays

- Define rays by an initial point and a direction: $\mathbf{x}(t)=\mathbf{x}_0+t\mathbf{d}$
- Eye rays: Rays from the eye through a pixel
  - Construct using the eye location and the pixel's location on the image plane. $\mathbf{X}_0$ = **eye**
- Shadow rays: Rays from a point on a surface to the light.
  - $\mathbf{X}_0$ = point on surface
- Reflection rays: Rays from a point on a surface in the reflection direction
  - Construct using laws of reflection. $\mathbf{X}_0$ = surface point
- Transmitted rays: Rays from a point on a transparent surface through the surface
  - Construct using laws of refraction. $\mathbf{X}_0$ = surface point

From Pixels to Rays



$$\vec{u} = \frac{look \times up}{|look \times up|}$$

$$\vec{v} = \frac{look \times \vec{u}}{|look \times \vec{u}|}$$

$$\Delta\vec{x} = \frac{2\tan(fov_x/2)}{W}\vec{u}$$

$$\Delta\vec{y} = \frac{2\tan(fov_y/2)}{H}\vec{v}$$

$$\vec{d}(i,j) = \frac{look}{|look|} + \frac{(2i+1-W)}{2}\Delta\vec{x} + \frac{(2j+1-H)}{2}\Delta\vec{y}$$

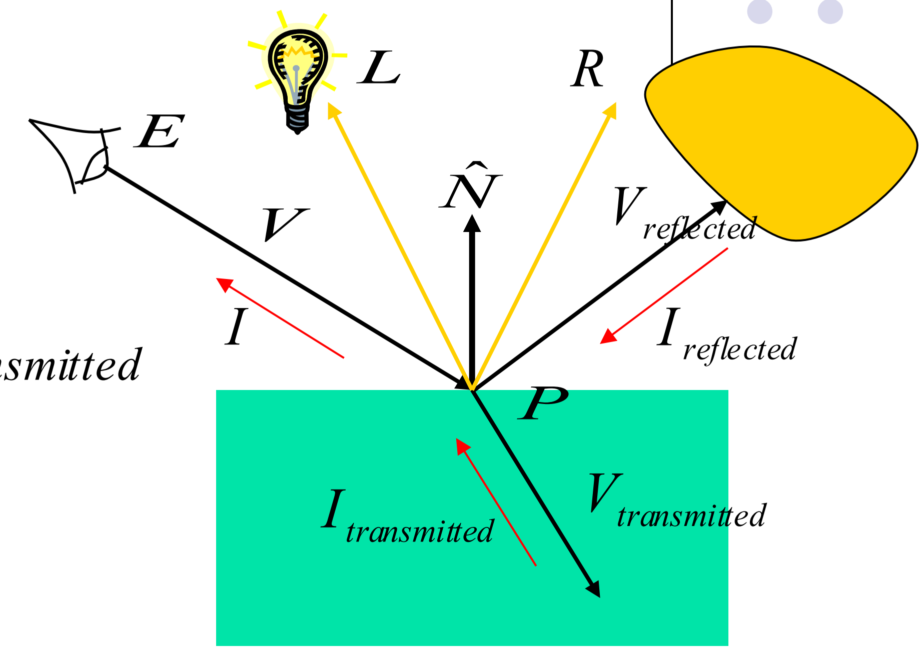
Ray Tracing Illumination

Recursive

$$I(E,V) = I_{direct} + I_{reflected} + I_{transmitted}$$

$$I_{reflected} = k_r I(P, V_{reflected})$$

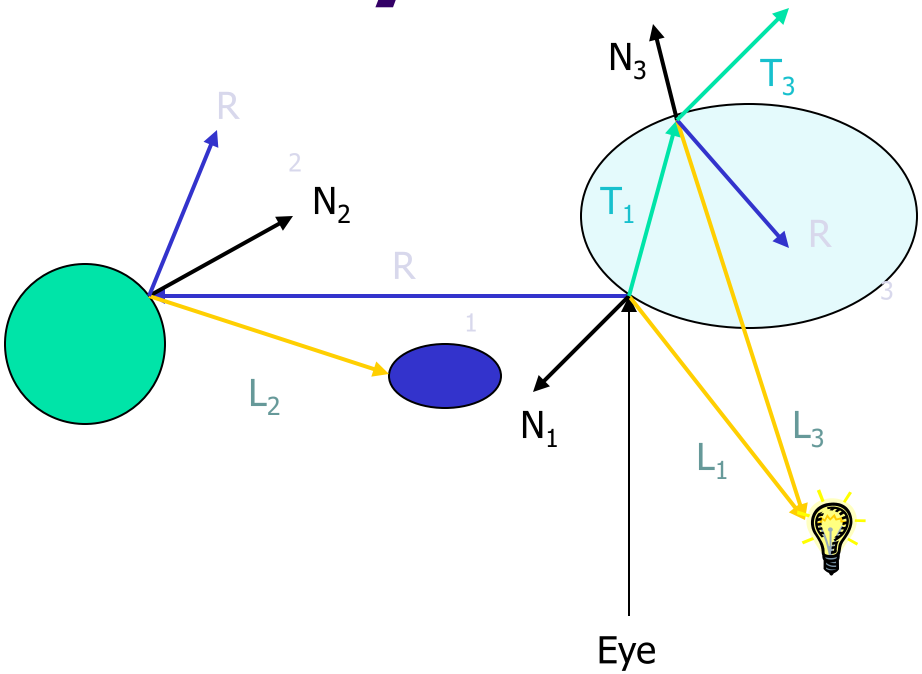
$$I_{transmitted} = k_t I(P, V_{transmitted})$$

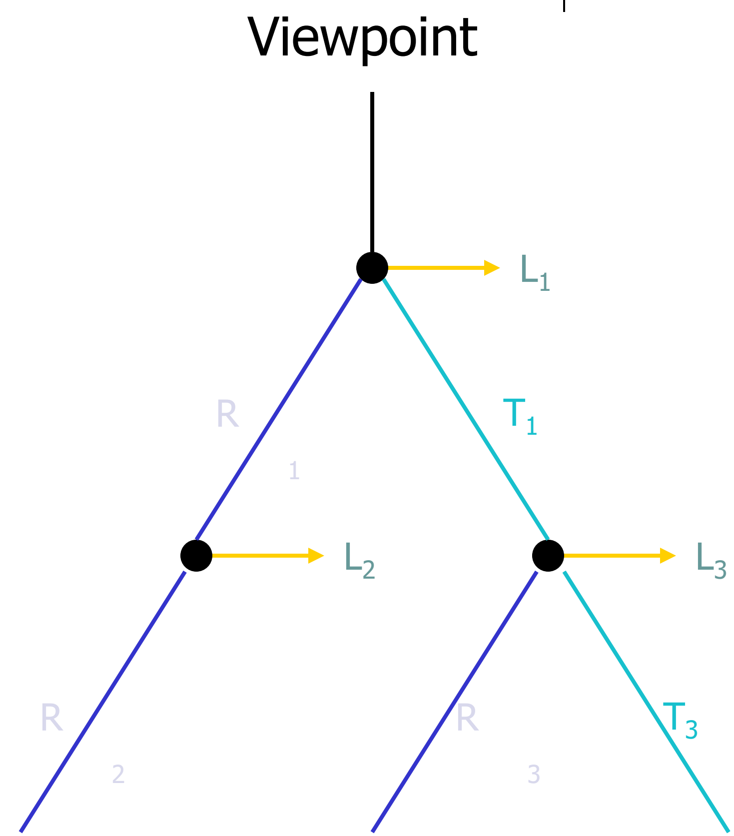


$$I_{direct} = k_a I_{ambient} + I_{light}\left[k_d\left(\hat{N}\cdot\hat{L}\right) + k_s\left(-\hat{V}\cdot\hat{R}\right)^{n_{shiny}}\right]$$

Check for shadowing (intersection with object along ray (P,L))

The Ray Tree





N_i surface normal

R_i reflected ray

L_i shadow ray

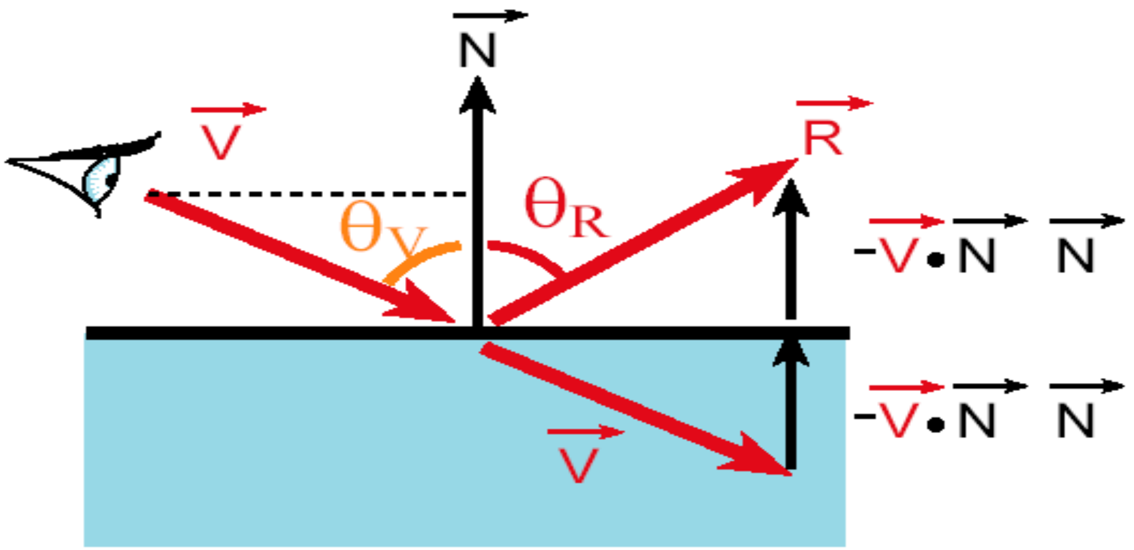
T_i transmitted (refracted) ray

Psuedo-code

Reflection

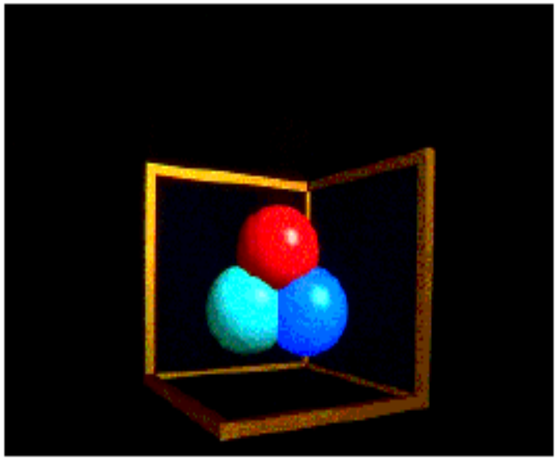
- Reflection angle = view angle

$$\vec{R} = \vec{V} - 2\left(\vec{V} \bullet \vec{N}\right)\vec{N}$$

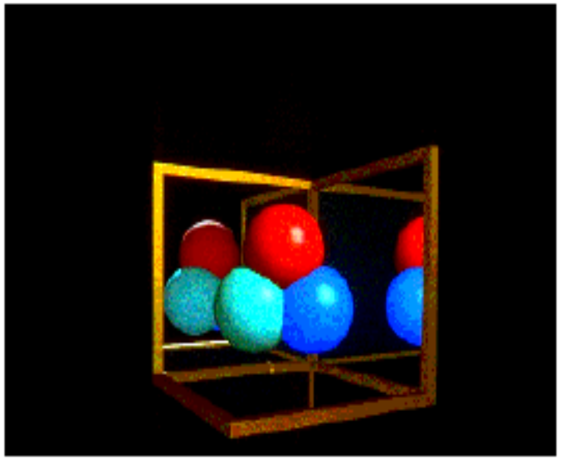


Reflection

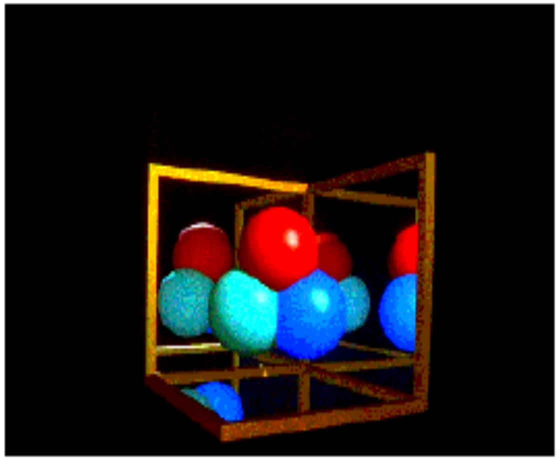
- The maximum depth of the tree affects the handling of refraction
- If we send another reflected ray from here, when do we stop? 2 solutions (complementary)
  - Answer 1: Stop at a fixed depth.
  - Answer 2: Accumulate product of reflection coefficients and stop when this product is too small.



0 recursion

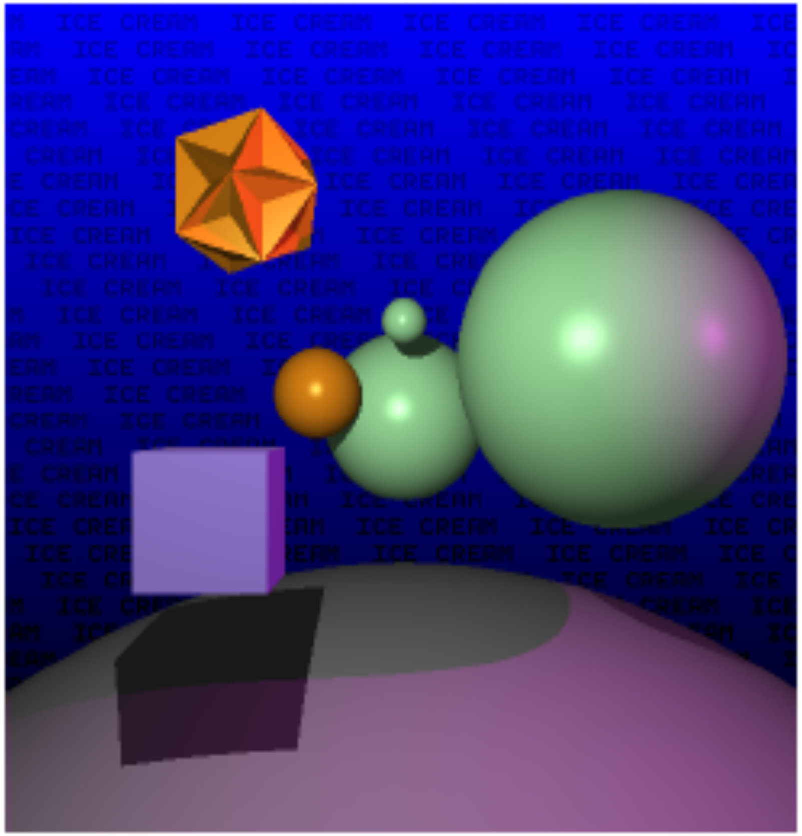


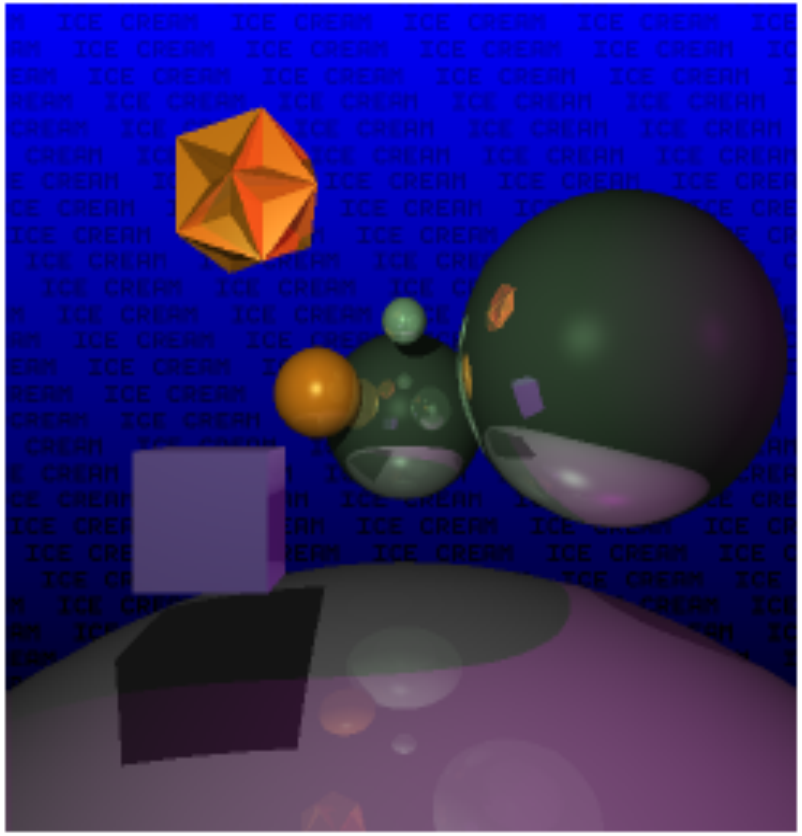
1 recursion



2 recursions

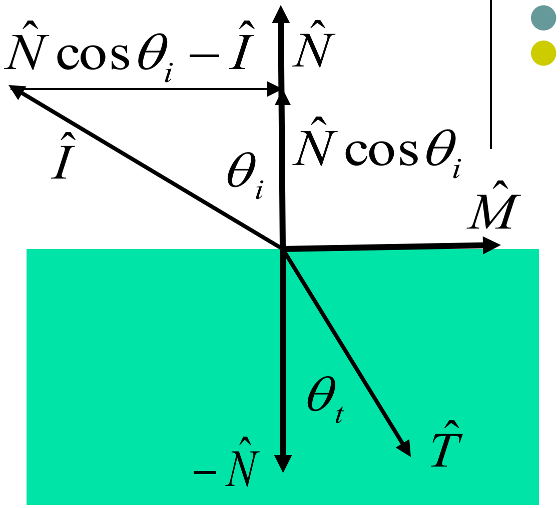
Reflection





Refraction

Snell's Law $\frac{\sin\theta_t}{\sin\theta_i} = \frac{\eta_i}{\eta_t} = \eta_r$



Note that I is the negative of the incoming ray

Pseudo Code for Ray Tracing

```
rgb lsou;        // intensity of light source
rgb back;        // background intensity
rgb ambi;        // ambient light intensity

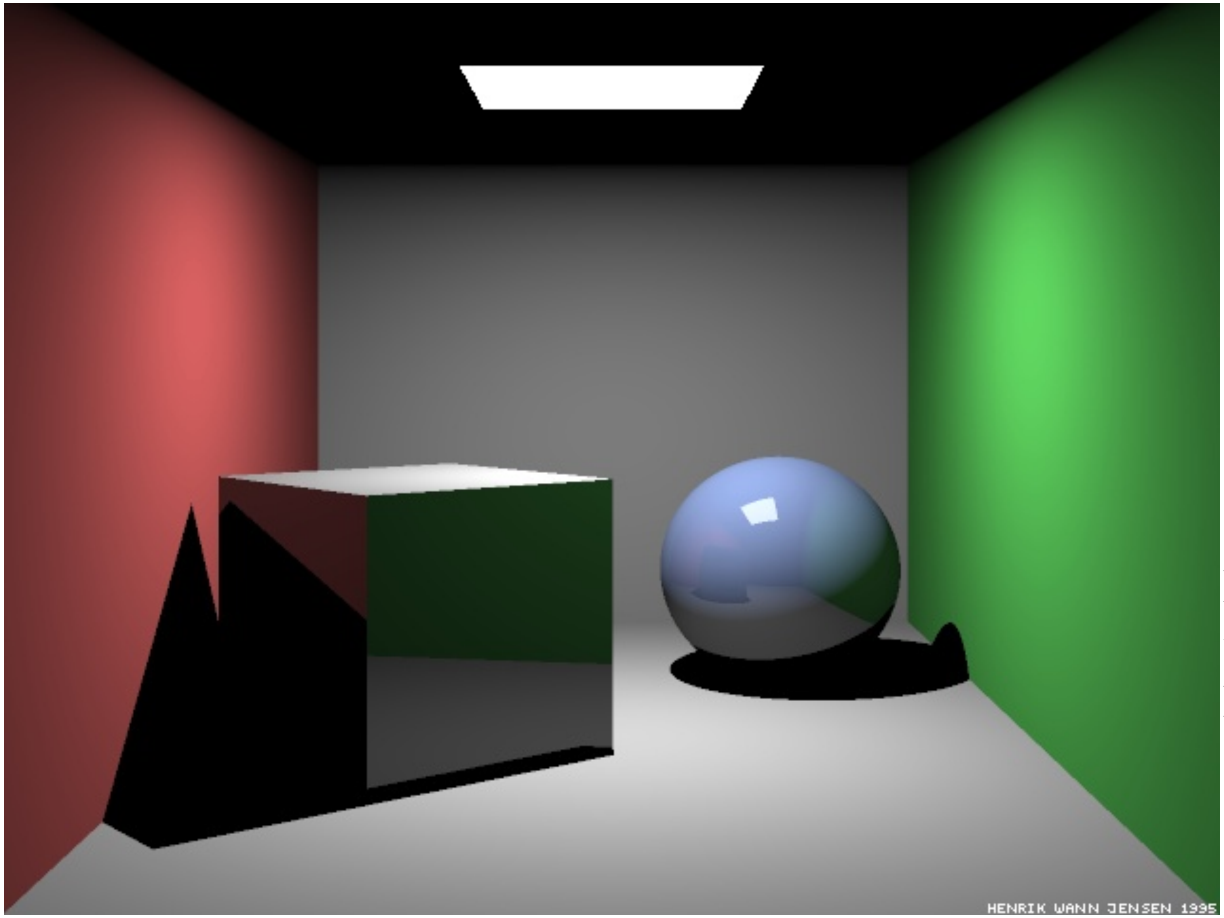
Vector L         // vector pointing to light source
Vector N         // surface normal
Object objects [n] //list of n objects in scene
float Ks [n]     // specular reflectivity factor for each object
float Kr [n]     //  refractivity index for each object
float Kd [n]     // diffuse reflectivity factor for each object
Ray r;

void raytrace() {
   for (each pixel P of projection viewport in raster order) {
        r = ray emanating from viewer through P
        int depth = 1; // depth of ray tree consisting of multiple paths
        the pixel color at P = intensity(r, depth)
   }
}
```

```
rgb intensity (Ray r, int depth) {
   Ray  flec, frac;
   rgb   spec, refr, dull, intensity;

    if (depth >= 5) intensity = back;
    else {
        find the closest intersection of r with all objects in scene
        if (no intersection) {
            intensity =back;
        } else {
            Take closest intersection which is object[j]
            compute normal N at the intersection point
            if (Ks[j] >0)   {  // non-zero specular reflectivity
                  compute reflection ray flec;
                  refl = Ks[j]*intensity(flec, depth+1);
            } else refl =0;
            if (Kr[j]>0)  {     // non-zero refractivity
                 compute refraction ray frac;
                 refr = Kr[j]*intensity(frac, depth+1);
            } else refr =0;
            check for shadow;
            if (shadow) direct = Kd[j]*ambi
            else direct = Phong illumination computation;
            intensity = direct + refl +refr;
       } }
     return intensity; }
```

Raytraced Cornell Box



Which paths are missing?

Ray-traced Cornell box, due to Henrik Jensen, http://www.gk.dtu.dk/~hwj

Paths in RayTracing

- Ray Tracing
  - Captures LDS*E paths: Start at the eye, any number of specular bounces before ending at a diffuse surface and going to the light
- Raytracing cannot do:
  - LS^*D^+E: Light bouncing off a shiny surface like a mirror and illuminating a diffuse surface
  - LD^+E: Light bouncing off one diffuse surface to illuminate others
- Basic problem: The raytracer doesn't know where to send rays out of the diffuse surface to capture the incoming light
- Also a problem for rough specular reflection
  - Fuzzy reflections in rough shiny objects
- Need other rendering algorithms that get more paths

A Better Rendered Cornell Box