Ray Tracing in (Less than) One Weekend with Dlang

Social: @MichaelShah
Web: mshah.io
Courses: courses.mshah.io
YouTube: www.youtube.com/c/MikeShah

Presentor: Mike Shah, Ph.D.

13:30-14:15, Tue, August, 2 2022

45 minutes | Introductory Audience

Please do not redistribute slides without prior permission.

Goal(s) for today

What You're Going to Learn Today? (1/2)

- Today you're going to learn about building a Ray Tracer in the D Programming Language
    - The book to the right, written by Peter Shirley, has helped get many folks get introduced and started in the graphics industry.

What You’re Going to Learn Today? (2/2)

- To the D Language experts in the room, I don’t think I have any ‘amazing’ DLang tricks to show you.
  - However, I hope you will perhaps use this as inspiration to try a short project in D using your expertise
  - Every time I build a Ray Tracer I learn something new!
  - Otherwise, sit back and remember why you love D in this tour!
- For beginners (the audience for this talk) -- I hope this will serve as an excellent project to learn D

Why Ray Tracers?

- Fun to build--can be quite compact and short project--you can actually finish the project!
  - (or you can make a career out of it!)
- A ray tracer is an excellent project when a student asks ‘what project should I build to practice skills?’
  - I think they are a great project for also learning a new language
- My claim (and what I am going to show off) is that using the D Programming Language, you can build a Ray Tracer in well under 24 hours
  - It will be a delightful experience and help showcase D Lang as a language for software engineers.

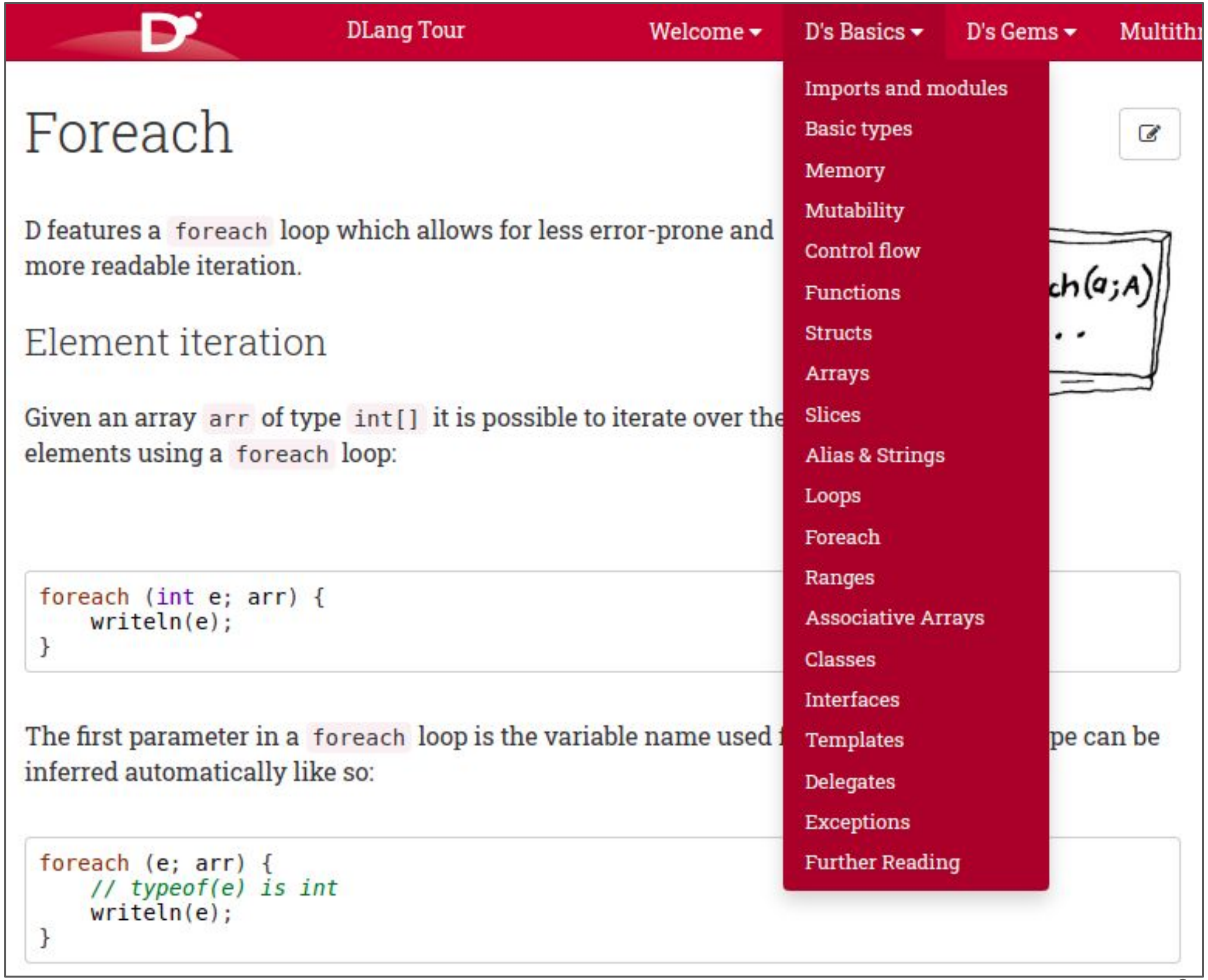
Your Guide for Today

by Mike Shah

- Associate Teaching Professor at Northeastern University in Boston, Massachusetts.
  - I teach courses in computer systems, computer graphics, and game engine development.
  - My research in program analysis is related to performance building static/dynamic analysis and software visualization tools.
- I do consulting and technical training on modern C++, Concurrency, OpenGL, and Vulkan projects
  - (Usually graphics or games related)
- I like teaching, guitar, running, weight training, and anything in computer science under the domain of computer graphics, visualization, concurrency, and parallelism.
- Contact information and more on: www.mshah.io
- More online training coming at courses.mshah.io

For Folks Online...

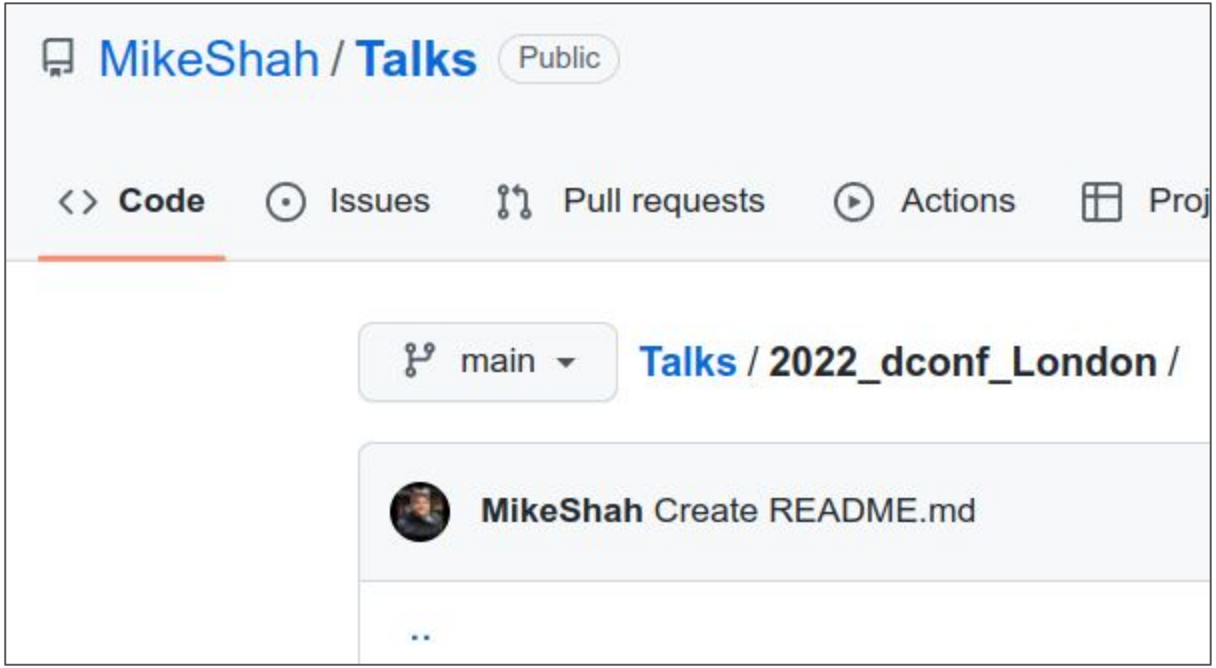
- I'm admitting I'm not a Dlang expert
  - (Please don't ask me to do live template metaprogramming examples during this talk :))
- The good news though--if you can walk through the 'D Basics' guide you can build a Ray Tracer



https://tour.dlang.org/

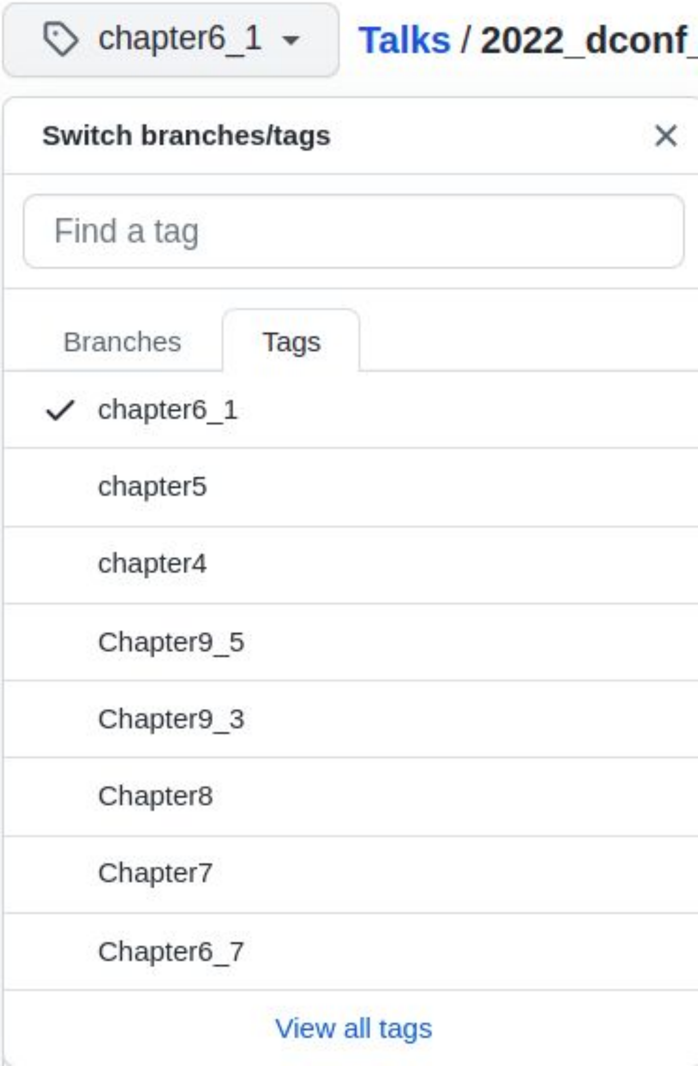
Code for the talk

- Located here: https://github.com/MikeShah/Talks/tree/main/2022_dconf_London



Code for the talk

- There are some tags on the github repository, roughly corresponding to the chapters in Shirleys text
  - Note: I'll start cleaning up the code more and more in the later tags. :)



Abstract

The abstract that you read and enticed you to join me is here!

Peter Shirley's book 'Ray Tracing in One Weekend' has been a brilliant introduction to implementing ray tracers for beginners. While you may not have read the book, you may have seen the viral images of programmers all over the world sharing images generated from their ray tracers on social media. While the book is implemented in C++, I was up for the challenge to implement the ray tracer in D. With the efficiency of the D language however, **I set myself the challenge to complete the task in less than 24 hours**. In this talk I will describe my experience, highlighting features of the D language that made this possible. I will also discuss why D may be the right language to teach software engineering in a university setting for this reason.

Ray Tracing

What does ‘ray tracing produce’

http://www.realtimerendering.com/raytracinggems/rtg/index.html

https://graphics.pixar.com/library/RayTracingCars/paper.pdf

Polygon

A scene from a Star Wars real-time ray tracing demo built in the Unreal Engine. | ILMxLAB

https://www.polygon.com/e3/2019/6/10/18410733/ray-tracing-xbox-scarlett-ps5-amd-navi-nvidia-rtx

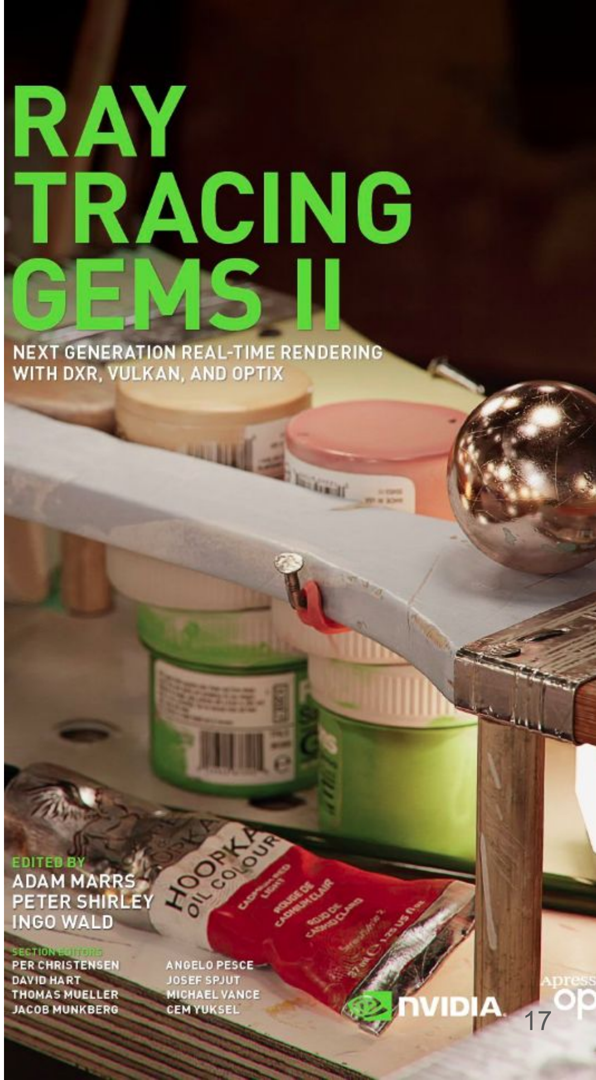

RTX
OFF

Ray tracing is a method of graphics rendering that simulates the physical behavior of light. Thought to be decades away from reality, NVIDIA has made real-time ray tracing possible with NVIDIA RTX™ the first-ever real-time ray-tracing GPU—and has continued to pioneer the technology since. Powered by NVIDIA RT Cores, ray tracing adds unmatched beauty and realism to renders and fits readily into preexisting development pipelines.

https://developer.nvidia.com/rtx/ray-tracing

The Ray Tracing Algorithm

(At a high level)

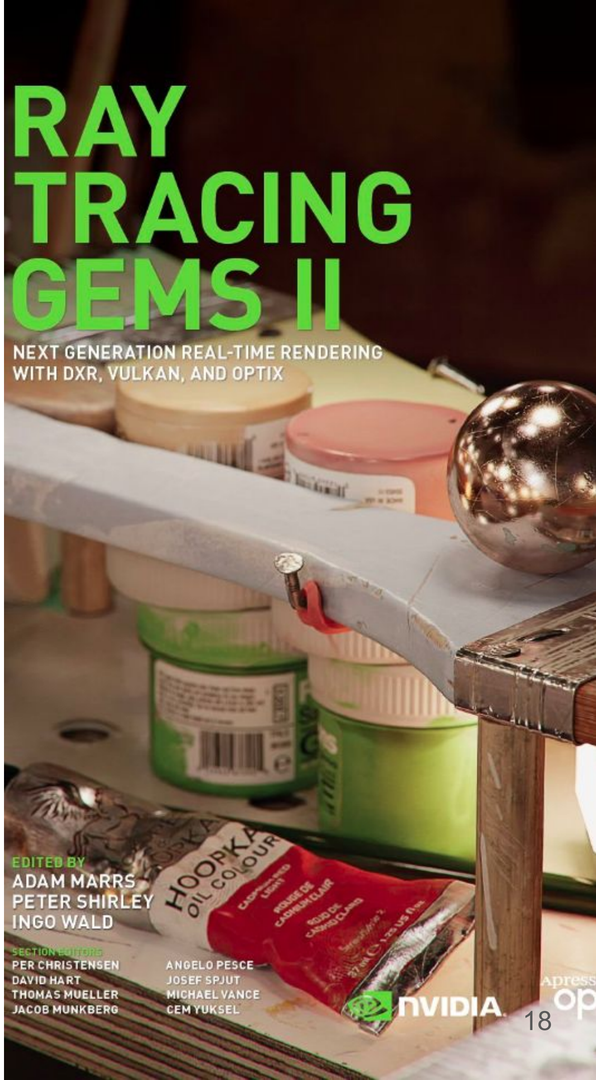


http://www.realtimerendering.com/raytracinggems/rtg/index.html

The Ray Tracing Algorithm

(At a high level)

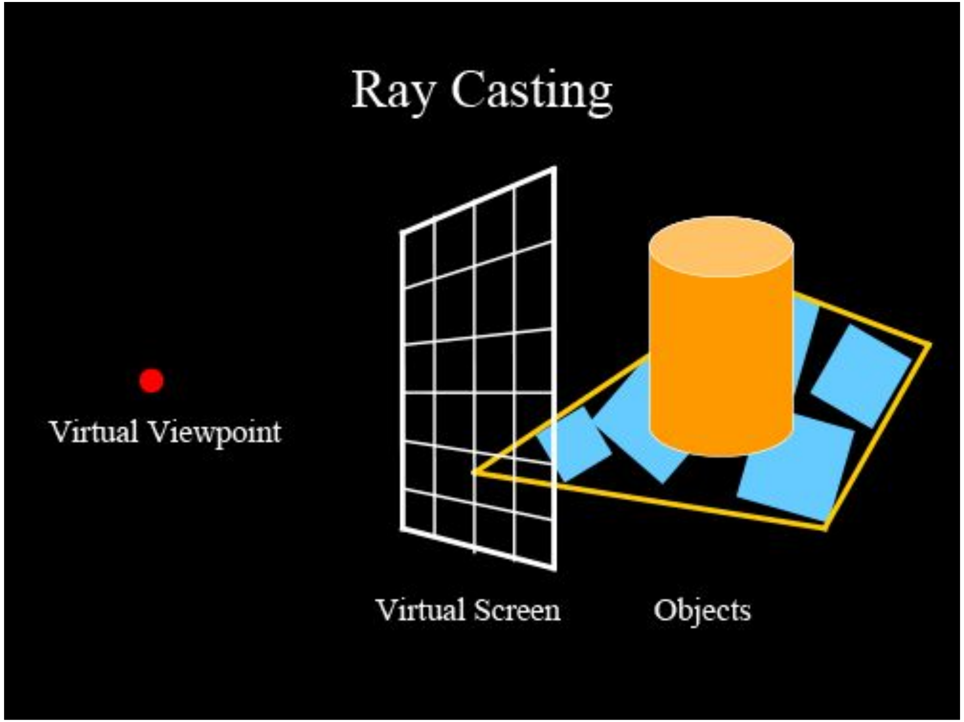
Let’s understand what we’re going to build at a high level, then we’ll start putting together the program.



http://www.realtimerendering.com/raytracinggems/rtg/index.html

Ray Tracing

- Ray Tracing is sort of what it sounds like.
- You are going to ‘cast’ a **ray** from an **origin** in some **direction**
  - And what we care about is where that ‘ray’ hits.
  - The intersection of a ray with an object is what gives us information about how to color in our scene.



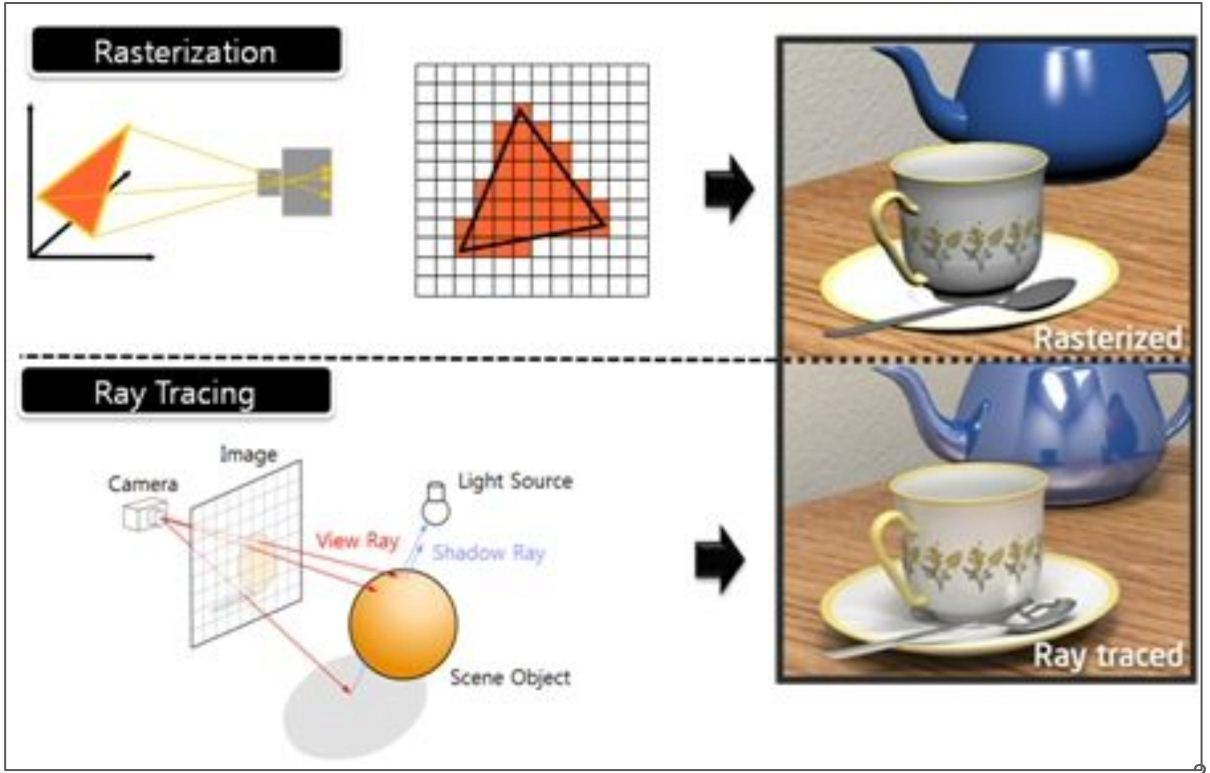
(Note: We’re doing ‘backwards ray tracing today, as we are casting the ray from a camera, rather than a light source)

Ray Tracing - Analogy

- The analogy is exactly like pointing a laser pointer
  - Our laser pointer hits the closest surface that it hits against
  - (The actual light particles may bounce multiple times, but we'll get into that).

Ray Tracing Versus Rasterization Algorithm

- Now Ray Tracing differs a bit from Rasterization
    - Rasterization is another technique for drawing 3D graphics scenes.
- Take a moment to see the difference between the two ideas and the generated images.



Building a Ray Tracer Code Examples

(C and Python)

A few code examples we can learn from

Ray Tracing in C (For compiled language performance) (1/2)

- Here's an example of a Ray Tracer in C
  - (I didn't write this one -- it is famously on the back of a business card by Andrew Kensler and there's another by Paul Heckbert in 1987)

Andrew Kensler's card - https://fabiensanglard.net/postcard_pathtracer/

Ray Tracers can be quite compact -- just a few core ideas to understand (1/2)

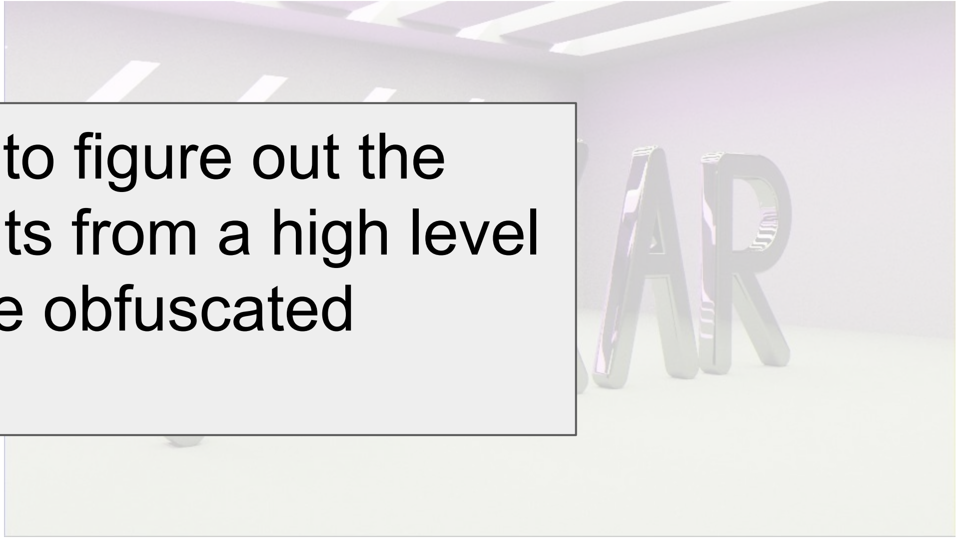
```
#include <stdlib.h> // card > pixar.ppm
#include <stdio.h>
#include <math.h>
#define R return
#define O operator
typedef float F;typedef int I;struct V{F x,y,z;V(F v=0){x=y=z=v;}V(F a,F b,F
c=0){x=a;y=b;z=c;}V O+(V r){R V(x+r.x,y+r.y,z+r.z);}V O*(V r){R V(x*r.x,y*r.
y,z*r.z);}F O%(V r){R x*r.x+y*r.y+z*r.z;}V O!(){R*this*(1/sqrtf(*this%*this)
);}};F L(F l,F r){R l<r?l:r;}F U(){R(F)rand()/RAND_MAX;}F B(V p,V l,V h){l=p
+l*-1;h=h+p*-1;R-L(L(L(l.x,h.x),L(l.y,h.y)),L(l.z,h.z));}F S(V p,I&m){F d=1\
e9;V f=p;f.z=0;char l[]="5O5_5W9W5_9_COC_AOEOA_E_IOQ_I_QOUOY_Y_]OWW[WaOa_aW\
eWa_e_cWiO";for(I i=0;i<60;i+=4){V b=V(l[i]-79,l[i+1]-79)*.5,e=V(l[i+2]-79,l
[i+3]-79)*.5+b*-1,o=f+(b+e*L(-L((b+f*-1)%e/(e%e),0),1))*-1;d=L(d,o%o);}d=sq\
rtf(d);V a[]={V(-11,6),V(11,6)};for(I i=2;i--;){V o=f+a[i]*-1;d=L(d,o.x>0?f\
absf(sqrtf(o%o)-2):(o.y+=o.y>0?-2:2,sqrtf(o%o)));}d=powf(powf(d,8)+powf(p.z,
8),.125)-.5;m=1;F r=L(-L(B(p,V(-30,-.5,-30),V(30,18,30)),B(p,V(-25,17,-25),V
(25,20,25))),B(V(fmodf(fabsf(p.x),8),p.y,p.z),V(1.5,18.5,-25),V(6.5,20,25)))
;if(r<d)d=r,m=2;F s=19.9-p.y;if(s<d)d=s,m=3;R d;}I M(V o,V d,V&h,V&n){I m,s=
0;F t=0,c;for(;t<100;t+=c)if((c=S(h=o+d*t,m))<.01||++s>99)R n=!V(S(h+V(.01,0
),s)-c,S(h+V(0,.01),s)-c,S(h+V(0,0,.01),s)-c),m;R 0;}V T(V o,V d){V h,n,r,t=
1,l(!V(.6,.6,1));for(I b=3;b--;){I m=M(o,d,h,n);if(!m)break;if(m==1){d=d+n*(
n%d*-2);o=h+d*.1;t=t*.2;}if(m==2){F i=n%l,p=6.283185*U(),c=U(),s=sqrtf(1-c),
g=n.z<0?-1:1,u=-1/(g+n.z),v=n.x*n.y*u;d=V(v,g+n.y*n.y*u,-n.y)*(cosf(p)*s)+V(
1+g*n.x*n.x*u,g*v,-g*n.x)*(sinf(p)*s)+n*sqrtf(c);o=h+d*.1;t=t*.2;if(i>0&&M(h
+n*.1,l,h,n)==3)r=r+t*V(500,400,100)*i;}if(m==3){r=r+t*V(50,80,100);break;}}
R r;}I main(){I w=960,h=540,s=16;V e(-22,5,25),g=!(V(-3,4,0)+e*-1),l=!V(g.z,
0,-g.x)*(1./w),u(g.y*l.z-g.z*l.y,g.z*l.x-g.x*l.z,g.x*l.y-g.y*l.x);printf("P\
6 %d %d 255 ",w,h);for(I y=h;y--;)for(I x=w;x--;){V c;for(I p=s;p--;)c=c+T(e
,!(g+l*(x-w/2+U())+u*(y-h/2+U())));c=c*(1./s)+14./241;V o=c+1;c=V(c.x/o.x,c.
y/o.y,c.z/o.z)*255;printf("%c%c%c",(I)c.x,(I)c.y,(I)c.z);}}// Andrew Kensler
```

Andrew Kensler's card - https://fabiensanglard.net/postcard_pathtracer/

Ray Tracers can be quite compact -- just a few core ideas to understand (2/2)

```
#include <stdlib.h> // card > pixar.ppm
#include <stdio.h>
#include <math.h>
#define R return
#define O operator
typedef float F;typedef int I;struct V{F
c=0){x=a;y=b;z=c;}V O+(V r){R V(x+r.x,y+
y,z*r.z);}F O%(V r){R x*r.x+y*r.y+z*r.z;
);}};F L(F l,F r){R l<r?l:r;}F U(){R(F)r
+l*-1;h=h+p*-1;R-L(L(L(l.x,h.x),L(l.y,h.
e9;V f=p;f.z=0;char l[]="5O5_5W9W5_9_COC
eWa_e_cWiO";for(I i=0;i<60;i+=4){V b=V(l
[i+3]-79)*.5+b*-1,o=f+(b+e*L(-L((b+f*-1)
rtf(d);V a[]={V(-11,6),V(11,6)};for(I i=
absf(sqrtf(o%o)-2):(o.y+=o.y>0?-2:2,sqrt
8),.125)-.5;m=1;F r=L(-L(B(p,V(-30,-.5,-
(25,20,25))),B(V(fmodf(fabsf(p.x),8),p.y
;if(r<d)d=r,m=2;F s=19.9-p.y;if(s<d)d=s,
0;F t=0,c;for(;t<100;t+=c)if((c=S(h=o+d*
),s)-c,S(h+V(0,.01),s)-c,S(h+V(0,0,.01),
1,l(!V(.6,.6,1));for(I b=3;b--;){I m=M(o,d,h,n);if(!m)break;if(m==1){d=d+n*(
n%d*-2);o=h+d*.1;t=t*.2;}if(m==2){F i=n%l,p=6.283185*U(),c=U(),s=sqrtf(1-c),
g=n.z<0?-1:1,u=-1/(g+n.z),v=n.x*n.y*u;d=V(v,g+n.y*n.y*u,-n.y)*(cosf(p)*s)+V(
1+g*n.x*n.x*u,g*v,-g*n.x)*(sinf(p)*s)+n*sqrtf(c);o=h+d*.1;t=t*.2;if(i>0&&M(h
+n*.1,l,h,n)==3)r=r+t*V(500,400,100)*i;}if(m==3){r=r+t*V(50,80,100);break;}}
R r;}I main(){I w=960,h=540,s=16;V e(-22,5,25),g=!(V(-3,4,0)+e*-1),l=!V(g.z,
0,-g.x)*(1./w),u(g.y*l.z-g.z*l.y,g.z*l.x-g.x*l.z,g.x*l.y-g.y*l.x);printf("P\
6 %d %d 255 ",w,h);for(I y=h;y--;)for(I x=w;x--;){V c;for(I p=s;p--;)c=c+T(e
,!(g+l*(x-w/2+U())+u*(y-h/2+U()))));c=c*(1./s)+14./241;V o=c+1;c=V(c.x/o.x,c.
y/o.y,c.z/o.z)*255;printf("%c%c%c",(I)c.x,(I)c.y,(I)c.z);}}// Andrew Kensler
```

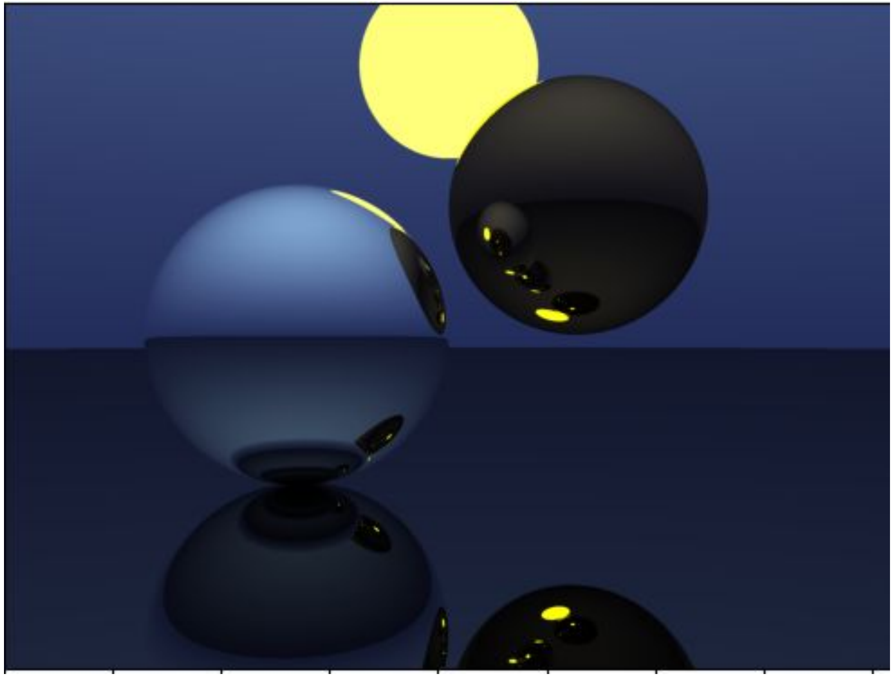
Let’s again try to figure out the key components from a high level (rather than the obfuscated source code)



Andrew Kensler’s card - https://fabiensanglard.net/postcard_pathtracer/

Python Ray Tracer (1/6)

- Here's an example of a Ray Tracer I wrote in Python 3.
  - It was again from a workshop taught by Peter Shirley (The 'author of Ray Tracing in One Weekend)

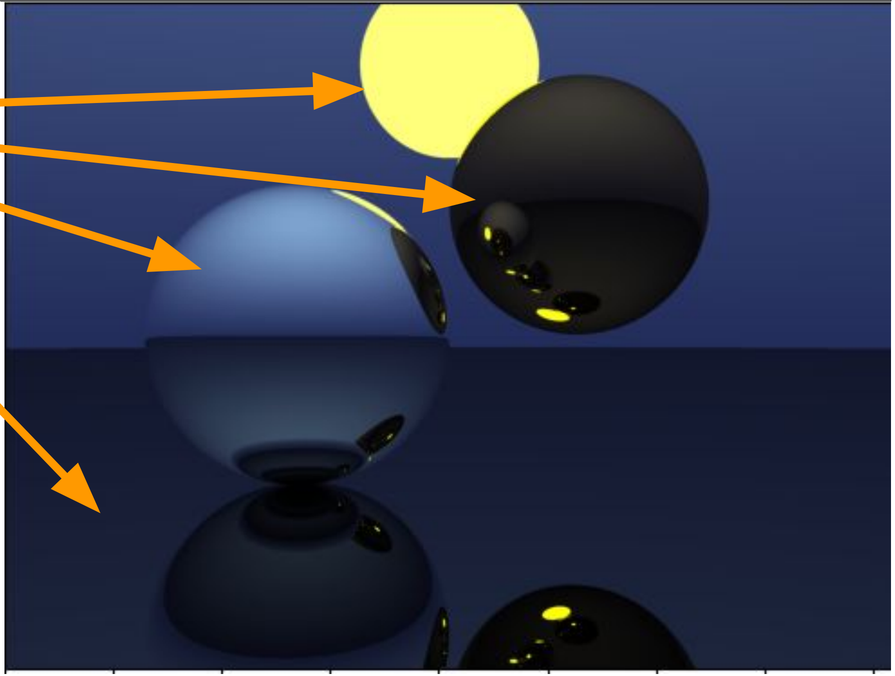


https://twitter.com/MichaelShah/status/1384621463018385415

Python Ray Tracer (2/6)

- Structurally:
  - We have a few objects

3 spheres, and 1 sphere on the ground

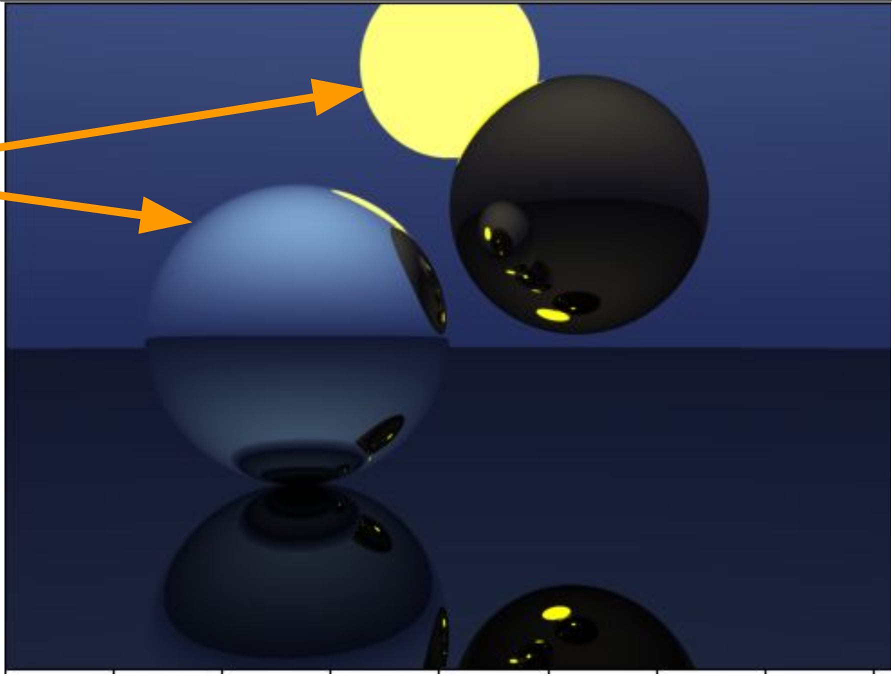


https://twitter.com/MichaelShah/status/1384621463018385415

Python Ray Tracer (3/6)

- Structurally:
  - We have a few objects
  - Each of these objects can be of a different material

Metal and another solid material (i.e. diffuse material)

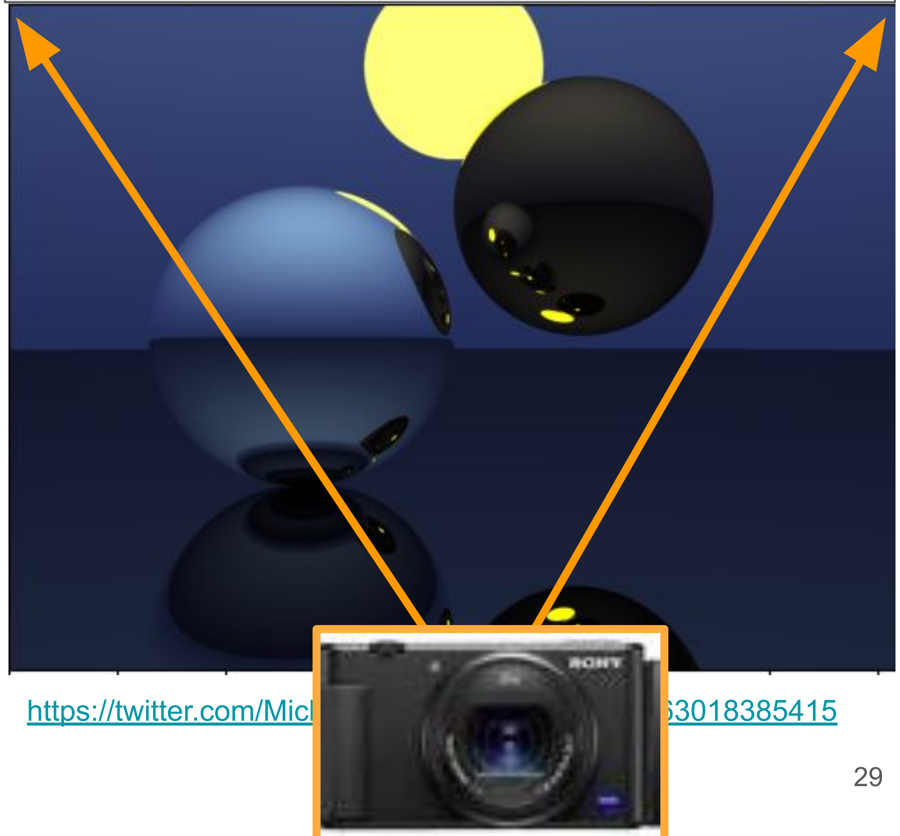


https://twitter.com/MichaelShah/status/1384621463018385415

Python Ray Tracer (4/6)

- Structurally:
  - We have a few objects
  - Each of these objects can be of a different material
  - We also have a camera viewing our scene from some perspective

We could have ‘one’ or more cameras

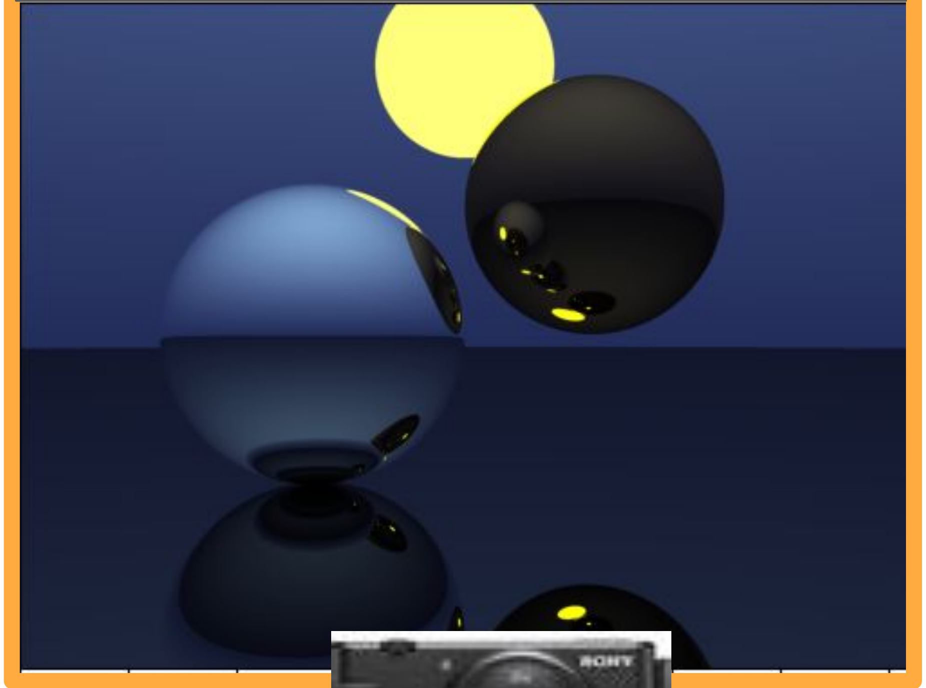


https://twitter.com/Micl … 63018385415

Python Ray Tracer (5/6)

- Structurally:
  - We have a few objects
  - Each of these objects can be of a different material
  - We also have a camera viewing our scene from some perspective
  - We also have the canvas where we are painting on as well!

The ‘canvas’ which we are painting on

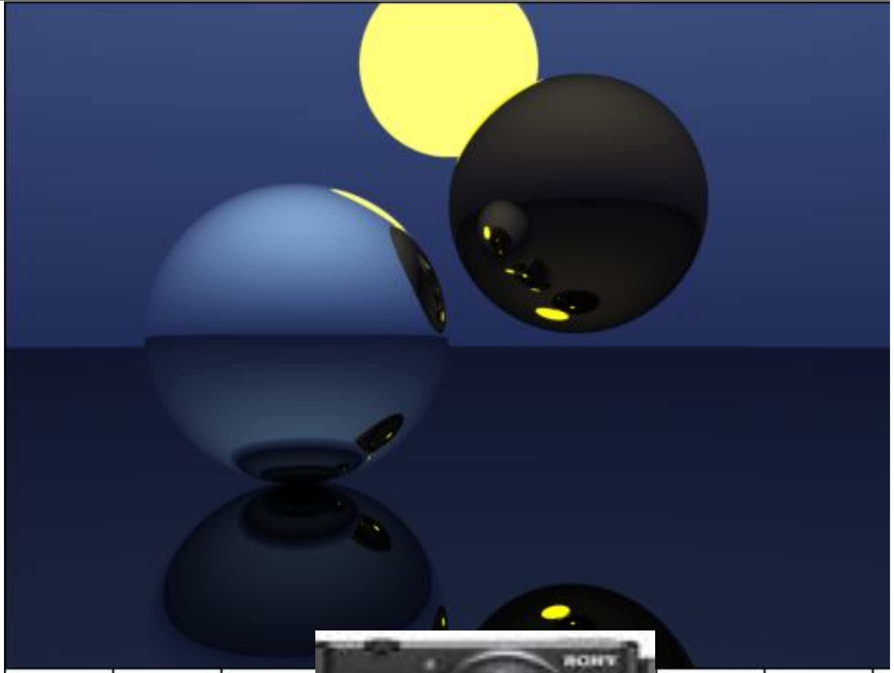


https://twitter.com/Mich 63018385415

Python Ray Tracer (6/6)

- Structurally:
  - We have a few objects
  - Each of these objects can be of a different material
  - We also have a camera viewing our scene from some perspective
  - We also have the canvas where we are painting on as well!
  - And finally, we're going to need some math
    - (Just a little bit, but you'll have to remember a few things from your high school geometry/algebra)

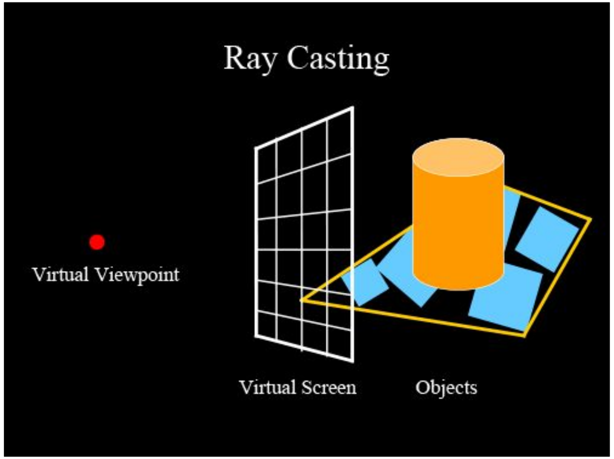
This is all done with mathematics! Students love graphics because they actually get to use nearly all the math they have learned -- they see the beauty!

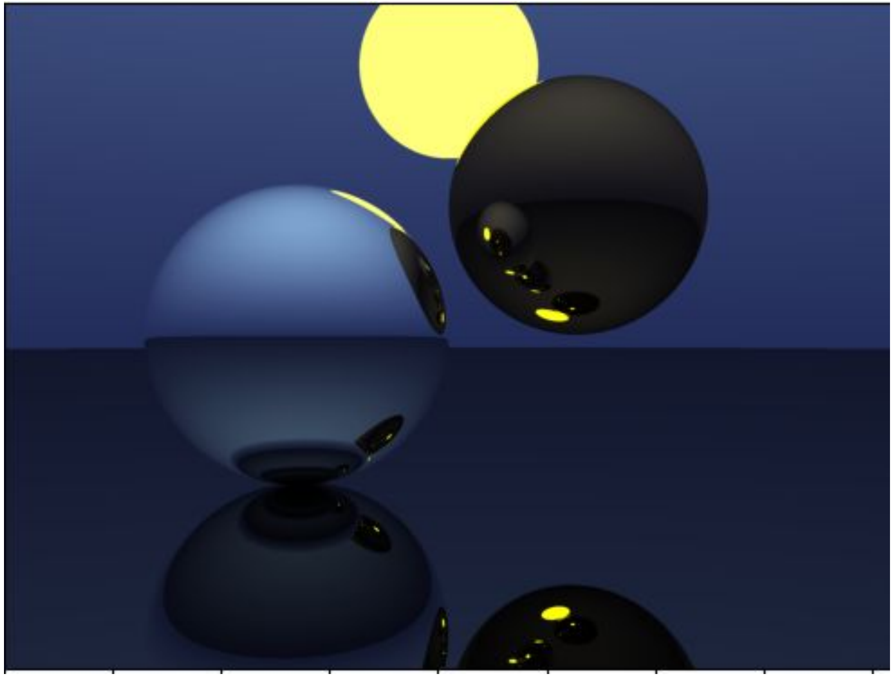


https://twitter.com/Mich 63018385415

Ray Tracer Algorithm Visualized (1/2)

- Now remember what we're doing, we're casting a 'ray' through one pixel at a time.

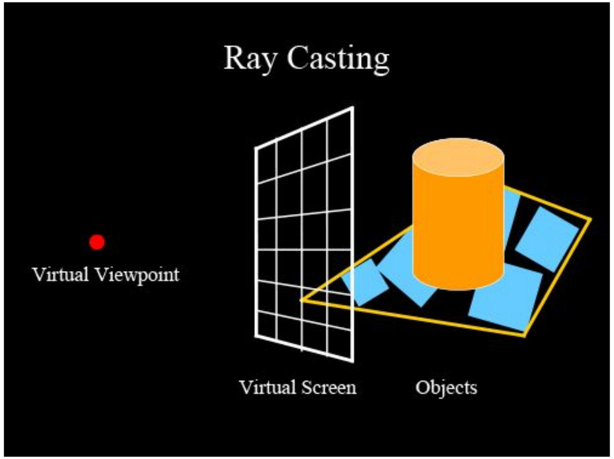




https://twitter.com/MichaelShah/status/1384621463018385415

Ray Tracer Algorithm Visualized (2/2)

- Now remember what we're doing, we're casting a 'ray' through one pixel at a time.
- To the right I've visualized the process (at a lower resolution)

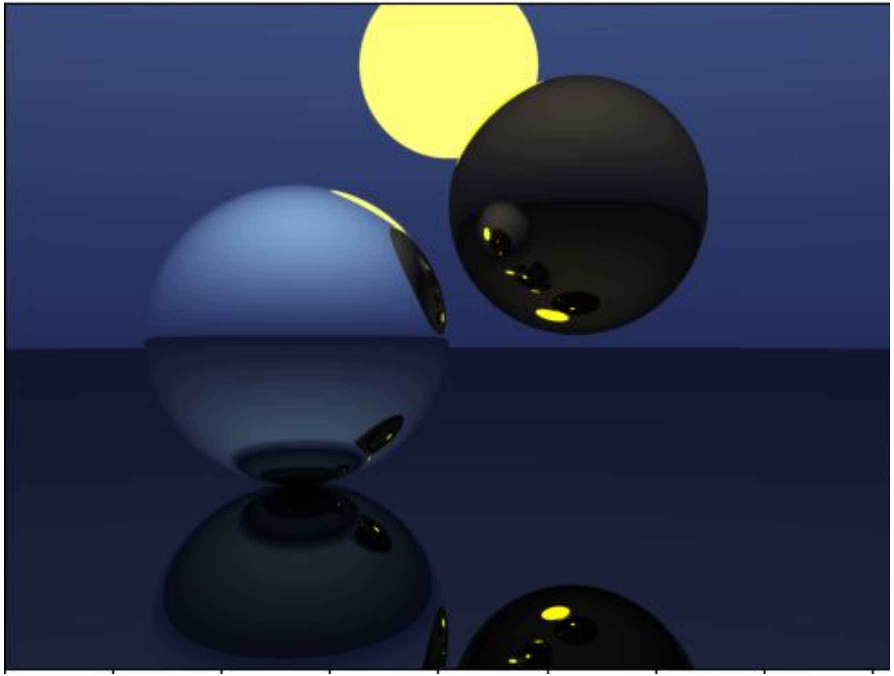


https://twitter.com/MichaelShah/status/1384621463018385415

Python Ray Tracer Code Sample (1/2)

(Look at a portion of the code here later if you like)

```
def hit_array(ray_origin, ray_direction, centers, radii):
    t_min = 9.0e8
    i_min = -1
    hit_something = False
    for i in range(len(centers)):
        t = hit_sphere(ray_origin, ray_direction, centers[i], radii[i])
        if (t >= 0.0001 and t < t_min):
            t_min = t
            i_min = i
            hit_something = True
    return (hit_something, t_min, i_min)

big_radius = 10000.0
radii = [ 1.0, big_radius, 1.0, 1.0 ]
centers = [ vec3(0.0, 3.0, -10.0), vec3(0.0, -big_radius - 1, 0.0), vec3(1.0, 1.0, -7.0), vec3(-1.0, 0.0, -6.0)]
for row in range(height):
   if (row % 50 == 0):
       print ('row = ', row)
   for column in range(width):
       u = (column + 0.5) / width
       v = (row + 0.5) / height
       ray_direction = unit_vector(vec3(2.0*u-1.0, aspect*(2.0*v-1.0), -window_depth))
       (hit_something, t, i) = hit_array(ray_origin, ray_direction, centers, radii)
       if (hit_something):
           hit_point = ray_origin + t*ray_direction
           surface_normal = (1.0/radii[i])*(hit_point - centers[i])
           im[height-row-1,column,:] = pseudocolor(surface_normal)
       else:
           im[height-row-1,column,:] = background_color( ray_direction )
im = im*im
plt.imshow(im)
plt.show()
```



Example of a final product.
https://twitter.com/MichaelShah/status/1384621463018385415

(This version was based off of Peter Shirley's Ray Tracing in 40 minutes Google Colab project -- very quick way to learn!)

Python R

(Look at a portio

```
def hit_array(ray_origin, ray_di
    t_min = 9.0e8
    i_min = -1
    hit_something = False
    for i in range(len(centers)):
        t = hit_sphere(ray_origi
        if (t >= 0.0001 and t <
            t_min = t
            i_min = i
            hit_something = T
    return (hit_something, t

big_radius = 10000.0
radii = [ 1.0, big_radius
centers = [ vec3(0.0, 3.
for row in range(height
   if (row % 50 == 0):
      print ('row =    ow)
   for column in ran e(width):
      u = (column + 0.5) / widt
      v = (row + 0.5) / height
      ray_direction = unit_vect
      (hit_something, t, i) = hit_array(ray_origin, ray_direction, centers, radii)
      if (hit_something):
         hit_point = ray_origin + t*ray_direction
         surface_normal = (1.0/radii[i])*(hit_point - centers[i])
         im[height-row-1,column,:] = pseudocolor(surface_normal)
      else:
         im[height-row-1,column,:] = background_color( ray_direction )
im = im*im
plt.imshow(im)
plt.show()
```

1. Iterate through each pixel one at a time
2. Cast a ray
3. Test if that ray hits something

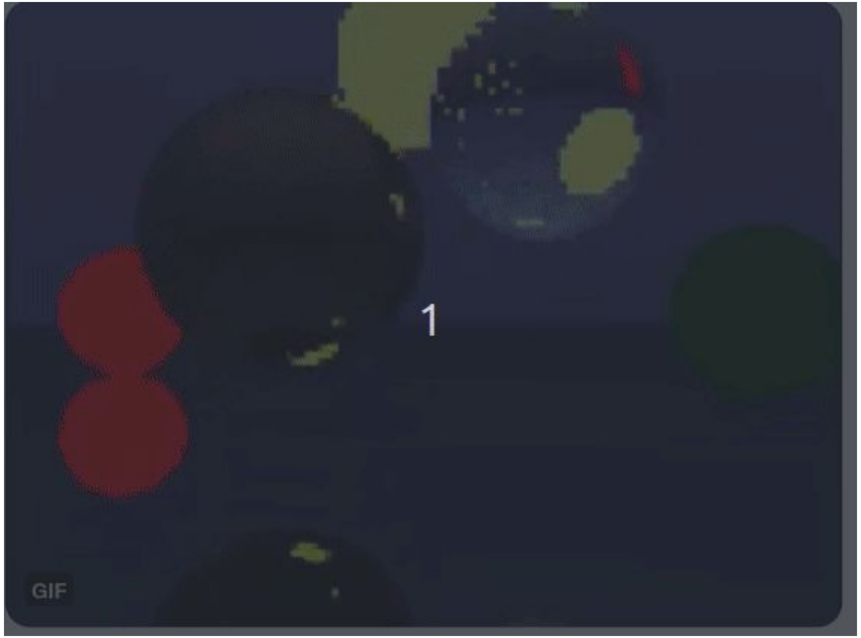
```
for row in range(height):
   if (row % 50 == 0):
       print ('row = ', row)
   for column in range(width):
       u = (column + 0.5) / width
       v = (row + 0.5) / height
       ray_direction = unit_vector(vec3(2.0*u-1.0, aspect*(2.0*v-1.0), -window_depth))
       (hit_something, t, i) = hit_array(ray_origin, ray_direction, centers, radii)
       if (hit_something):
           hit_point = ray_origin + t*ray_direction
           surface_normal = (1.0/radii[i])*(hit_point - centers[i])
           im[height-row-1,column,:] = pseudocolor(surface_normal)
       else:
           im[height-row-1,column,:] = background_color( ray_direction )
```

(This version was based off of Peter Shirley's Ray Tracing in 40 minutes Google Colab project -- very quick way to learn!)

Example of a final product.
https://twitter.com/MichaelShah/status/1384621463018385415

An Aside: Python Ray Tracer

- By altering our objects position, and re-rendering, we capture motion one frame after the other.
  - This is what we would do if we are making a computer generated film.
  - Add another loop to the previous snippet for ‘motion’ and you’re all set!



https://twitter.com/MichaelShah/status/1384621463018385415

Coming from C++ and Python **to!DLang**

- So I've implemented Ray Tracers in C++ and then Python
  - Now it's time to walk us through in Dlang
  - Now you know the main ingredients of a ray tracer, so you can follow along
- Dlang is a very productive language to program in
- Peter Shirley's [Ray Tracing in One Weekend] is your full guide written in C++
  - But we can follow along with Dlang nicely!.

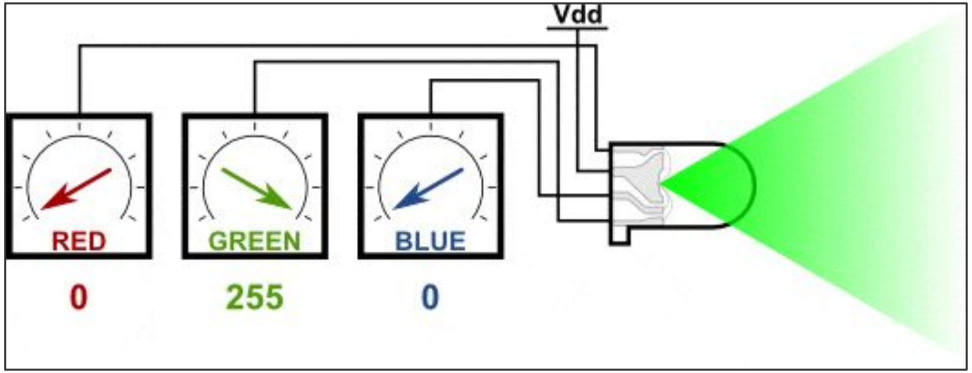
https://wiki.dlang.org/Coming_From

Let’s Begin our Journey Raytracing in D

The First Hour

Image Generation

- In order to start generating ‘pretty pictures’ we first have to store individual pixels to the screen
  - A pixel today means:
    - red, green, blue component at a specific position.
    - Typically these range from 0-255 for each color component
    - (Some systems use 0.0 to 1.0)
- We’ll also need need an image format to write out our canvas to save to disk.
  - The text-based PPM Format usually works quite well -- essentially raw data of pixels color components



PPM example [edit]

This is an example of a color RGB image stored in PPM format. There is a newline character at the end of each line.

```
P3
# The P3 means colors are in ASCII, then 3 columns and 2 rows,
# then 255 for max color, then RGB triplets
3 2
255
255   0   0     0 255   0     0   0 255
255 255   0   255 255 255     0   0   0
```

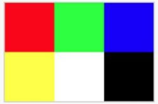


Image (magnified)

Figure 1: *PPM Example*

We are mapping our hardware to software -- that is actually pretty neat and a fun way to approach software engineering

From Peter Shirley (C++ Snippet)

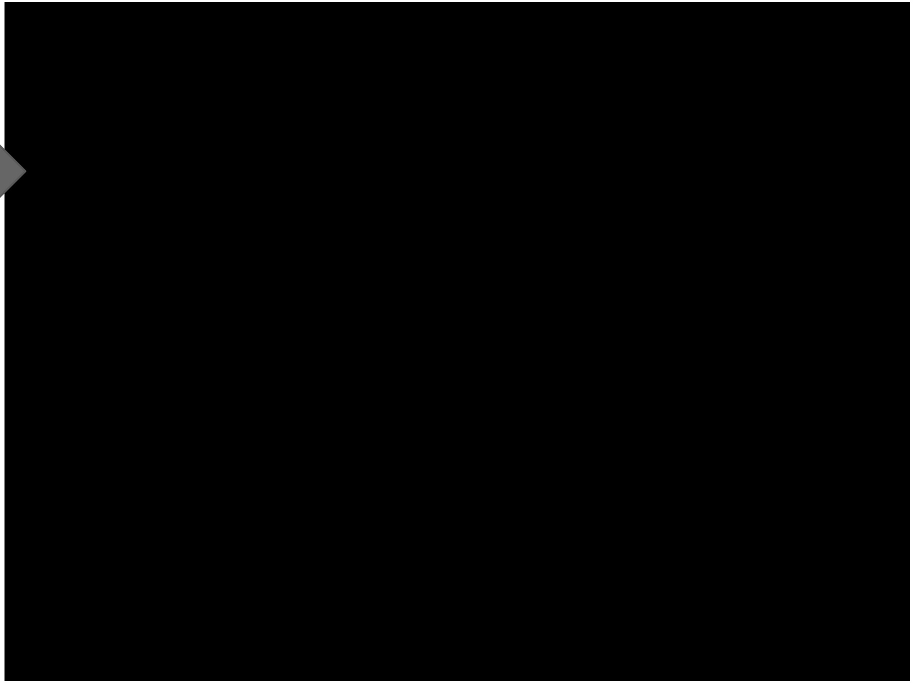
```cpp
#include <iostream>

int main() {

    // Image

    const int image_width = 256;
    const int image_height = 256;

    // Render

    std::cout << "P3\n" << image_width << ' ' << image_height << "\n255\n";

    for (int j = image_height-1; j >= 0; --j) {
        for (int i = 0; i < image_width; ++i) {
            auto r = double(i) / (image_width-1);
            auto g = double(j) / (image_height-1);
            auto b = 0.25;

            int ir = static_cast<int>(255.999 * r);
            int ig = static_cast<int>(255.999 * g);
            int ib = static_cast<int>(255.999 * b);

            std::cout << ir << ' ' << ig << ' ' << ib << '\n';
        }
    }
}
```

- The first goal is to just get something to show up on screen.
- I think this is a great way to approach the problem
    - Get something working
    - Then slowly iterate

From Peter Shirley (C++ Snippet) to D Code

```cpp
#include <iostream>

int main() {

    // Image

    const int image_width = 256;
    const int image_height = 256;

    // Render

    std::cout << "P3\n" << image_width << ' ' << image_height << "\n255\n";

    for (int j = image_height-1; j >= 0; --j) {
        for (int i = 0; i < image_width; ++i) {
            auto r = double(i) / (image_width-1);
            auto g = double(j) / (image_height-1);
            auto b = 0.25;

            int ir = static_cast<int>(255.999 * r);
            int ig = static_cast<int>(255.999 * g);
            int ib = static_cast<int>(255.999 * b);

            std::cout << ir << ' ' << ig << ' ' << ib << '\n';
        }
    }
}
```

C++ to Dlang



(Recording yourself coding in one take was...fun:))

Type Casts

- You might have caught I did some type casting from C++ static_cast here.
  - This casting mechanism actually becomes very important in Ray Tracers
    - Ray Tracers are very prone to overflow, or producing NaN values
      - (It's not obvious now though)

```
int ir = to!int(255.999 * r);
int ig = to!int(255.999 * g);
int ib = to!int(255.999 * b);
```

std.conv

A one-stop shop for converting values from one type to another.

Category	Functions
Generic	asOriginalType castFrom parse to toChars
Strings	text wtext dtext hexString
Numeric	octal roundTo signed unsigned
Exceptions	ConvException ConvOverflowException

Adding Abstraction - Our Canvas

Hours 2-4

The ‘canvas’ which we are painting on

A Canvas

- I need some way to write and store pixel information over time
  - Abstraction can be useful to give us access to individual pixels
  - So I’m going to write a ‘PPM’ class
  - DLang allows me to work in multiple paradigms (functional, procedural, and object-oriented)

Classes

D provides support for classes and interfaces like in Java or C++.

Any `class` type inherits from `Object` implicitly.

```
class Foo { } // inherits from Object
class Bar : Foo { } // Bar is a Foo too
```

Classes in D are generally instantiated on the heap using `new`:

```
auto bar = new Bar;
```

Class objects are always reference types and unlike `struct` aren't copied by value.

```
Bar bar = foo; // bar points to foo
```

The garbage collector will make sure the memory is freed when no references to an object exist anymore.

https://tour.dlang.org/tour/en/basics/classes

A Canvas - PPM Class (1/6)

- Here's a snapshot of the important details when building a class

```d
// ppm.d
module ppm;

import std.stdio;
import std.conv;

class PPM{
    /// Enums for PPM class for magic numbers
    enum MagicNumber {P3 = "P3", P6 = "P6"}

    /// Constructor for PPM image
    this(uint width, uint height){
        m_width = width;
        m_height = height;
        m_pixels = new ubyte[width*height*3];
    }

    /// Write the file
    /// If 'flip' is toggled to true, then flips the PPM image
    void WriteFile(string filename, MagicNumber magicNumber, const bool flip){
        File file = File(filename, "w");

        // Write out the header P3 header
        file.writeln(magicNumber);
        file.writeln(to!string(m_width)~" "~to!string(m_height));
        file.writeln(to!string(m_maxValue));
```

...

```d
    // Some constant values for the image dimensions
    private ubyte[]  m_pixels;
    private string  m_magicNumber   = "P3";
    private uint    m_width         = 256;
    private uint    m_height        = 256;
    private uint    m_maxValue      = 255;
}
```

A Canvas - PPM Class (2/6)

- Constructors neatly named ‘this’
- No destructors (garbage collected by default [more])

```
// ppm.d
module ppm;

import std.stdio;
import std.conv;

class PPM{
    /// Enums for PPM class for magic numbers
    enum MagicNumber {P3 = "P3", P6 = "P6"}

    /// Constructor for PPM image
    this(uint width, uint height){
        m_width = width;
        m_height = height;
        m_pixels = new ubyte[width*height*3];
    }

    /// Write the file
    /// If 'flip' is toggled to true, then flips the PPM image
    void WriteFile(string filename, MagicNumber magicNumber, const bool flip){
        File file = File(filename, "w");

        // Write out the header P3 header
        file.writeln(magicNumber);
        file.writeln(to!string(m_width)~" "~to!string(m_height));
        file.writeln(to!string(m_maxValue));
```

...

```
    // Some constant values for the image dimensions
    private ubyte[]  m_pixels;
    private string  m_magicNumber   = "P3";
    private uint    m_width         = 256;
    private uint    m_height        = 256;
    private uint    m_maxValue      = 255;
}
```

A Canvas - PPM Class (3/6)

- Member variables (Fields), and individually I can specify protection level (private)
    - Otherwise public by default, which I believe is the right choice [see access control]

```d
// ppm.d
module ppm;

import std.stdio;
import std.conv;

class PPM{
    /// Enums for PPM class for magic numbers
    enum MagicNumber {P3 = "P3", P6 = "P6"}

    /// Constructor for PPM image
    this(uint width, uint height){
        m_width = width;
        m_height = height;
        m_pixels = new ubyte[width*height*3];
    }

    /// Write the file
    /// If 'flip' is toggled to true, then flips the PPM image
    void WriteFile(string filename, MagicNumber magicNumber, const bool flip){
        File file = File(filename, "w");

        // Write out the header P3 header
        file.writeln(magicNumber);
        file.writeln(to!string(m_width)~" "~to!string(m_height));
        file.writeln(to!string(m_maxValue));
```

...

```d
    // Some constant values for the image dimensions
    private ubyte[]  m_pixels;
    private string  m_magicNumber   = "P3";
    private uint    m_width         = 256;
    private uint    m_height        = 256;
    private uint    m_maxValue      = 255;
}
```

A Canvas - PPM Class (4/6)

- One more thing to point out about variables
  - They have defaults!
  - Can even query their properties (i.e. .init, .min, .max, [see more](#))

```d
// ppm.d
module ppm;

import std.stdio;
import std.conv;

class PPM{
    /// Enums for PPM class for magic numbers
    enum MagicNumber {P3 = "P3", P6 = "P6"}

    /// Constructor for PPM image
    this(uint width, uint height){
        m_width = width;
        m_height = height;
        m_pixels = new ubyte[width*height*3];
    }

    /// Write the file
    /// If 'flip' is toggled to true, then flips the PPM image
    void WriteFile(string filename, MagicNumber magicNumber, const bool flip){
        File file = File(filename, "w");

        // Write out the header P3 header
        file.writeln(magicNumber);
        file.writeln(to!string(m_width)~" "~to!string(m_height));
        file.writeln(to!string(m_maxValue));
```

...

```d
e constant values for the image dimensions
te[]  m_pixels;
         m_magicNumber   = "P3";
    private uint   m_width      = 256;
    private uint   m_height     = 256;
    private uint   m_maxValue   = 255;
}
```

A Canvas - PPM Class (5/6)

- I can pack everything away into a nice ‘module’
  - Makes compilation and worrying about ‘header’ files less of a problem.

```
// ppm.d
module ppm;

import std.stdio;
import std.conv;

class PPM{
    /// Enums for PPM class for magic numbers
    enum MagicNumber {P3 = "P3", P6 = "P6"}

    /// Constructor for PPM image
    this(uint width, uint height){
        m_width = width;
        m_height = height;
        m_pixels = new ubyte[width*height*3];
    }

    /// Write the file
    /// If 'flip' is toggled to true, then flips the PPM image
    void WriteFile(string filename, MagicNumber magicNumber, const bool flip){
        File file = File(filename, "w");

        // Write out the header P3 header
        file.writeln(magicNumber);
        file.writeln(to!string(m_width)~" "~to!string(m_height));
        file.writeln(to!string(m_maxValue));
```

...

```
    // Some constant values for the image dimensions
    private ubyte[]  m_pixels;
    private string  m_magicNumber   = "P3";
    private uint    m_width         = 256;
    private uint    m_height        = 256;
    private uint    m_maxValue      = 255;
}
```

A Canvas - PPM Class (6/6)

- Many facilities for working with files (even .json) in D's standard library (The standard library is called Phobos) as well.
    - File utilities will be very familiar to C, C++, Python, etc. programmers

```d
// ppm.d
module ppm;

import std.stdio;
import std.conv;

class PPM{
    /// Enums for PPM class for magic numbers
    enum MagicNumber {P3 = "P3", P6 = "P6"}
```

I/O & File system	[https://dlang.org/phobos/index.html]
std.file	Manipulate files and directories.
std.path	Manipulate strings that represent filesystem paths.
std.stdio	Perform buffered I/O.

```d
    WriteFile(string filename, MagicNumber magicNumber, const bool flip){
        File file = File(filename, "w");

        // Write out the header P3 header
        file.writeln(magicNumber);
        file.writeln(to!string(m_width)~" "~to!string(m_height));
        file.writeln(to!string(m_maxValue));
```

...

```d
    // Some constant values for the image dimensions
    private ubyte[]  m_pixels;
    private string   m_magicNumber   = "P3";
    private uint     m_width         = 256;
    private uint     m_height        = 256;
    private uint     m_maxValue      = 255;
}
```

Drawing spheres and rays on our canvas

Hours 4-10

We want to draw some shapes

Drawing Shapes (Requires Math)

- So in order to draw, we have to represent a circle (equation to the right)
- **Our goal** is to determine if a **ray** intersects this circle
  - If a ray crosses through 1 (or 2 times) then we show some color
  - If it does not, then the ray continues on its journey to see if it intersects with something else

$$(x - C_x)^2 + (y - C_y)^2 + (z - C_z)^2 = r^2$$

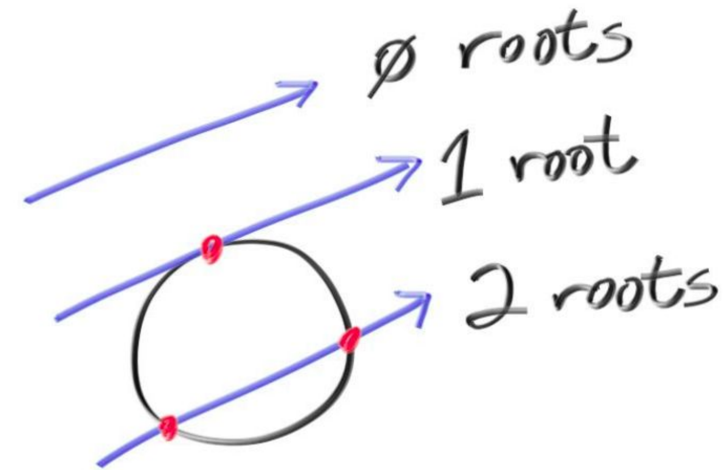
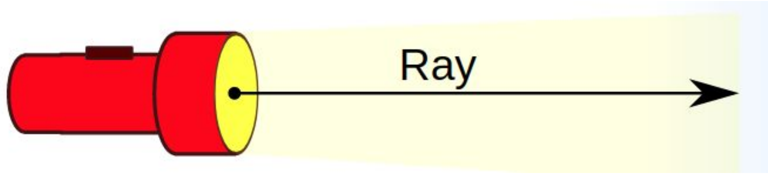


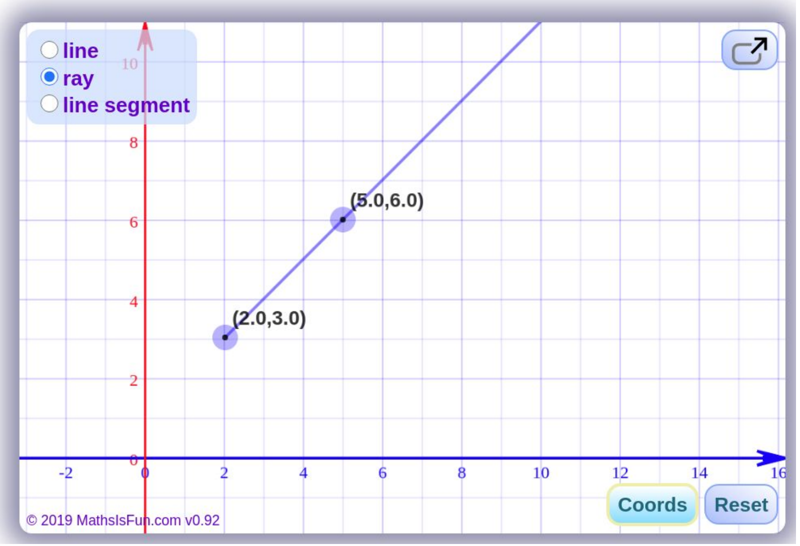
Figure 4: *Ray-sphere intersection results*

https://raytracing.github.io/books/RayTracingInOneWeekend.html#addingasphere

A Ray Class (1/2)

- Again, we're going to need some sort of abstraction for our 'Ray Class'
  - A Ray by definition has:
    - an **origin** (where we are holding our laser pointer)
    - a **direction** (where the ray extends out)





A Ray Class (2/2)

- Again, we're going to need some sort of abstraction for our 'Ray Class'
  - A Ray by definition has:
    - an **origin** (where we are holding our laser pointer)
    - a **direction** (where the ray extends out)

The direction itself can be represented by a 3D-vector.

Vec3 Class

- In its simplest form, a 3D vector stores 3 doubles.
  - See the example to the right
- (Note: DLang makes function/class/interface templates easy to implement -- see lower code listing)

```
class Vec3 (T){
    T[3] elements;
}

void main(){
    Vec3!double v1;
    Vec3!float v2;
}
```

```
module vec3;

class Vec3{
    import std.meta;
    import std.math;

    /// Constructor for a Vec3
    this(){
        e[0] = 0.0;
        e[1] = 0.0;
        e[2] = 0.0;
    }
```

Vec3/Point3/Color Class

- Vec3 will be used to represent things like point3, rgb, etc. in graphics programming.
  - DLang supports ‘alias’ as an easy way to provide some context. [[see alias](#)]
  - I find this extremely important for library writing
    - (Although, the non-expert in me wants to see if the alias can be enforced more strongly, maybe in a [precondition contract](#) -- to be returned to!)

```
// https://dlang.org/library/std/meta/alias.html
import std.meta;

class Vec3 (T){
    T[3] elements;
}

alias Vector3 = Vec3!double;
alias Point3  = Vec3!double;
alias rgb     = Vec3!double;

void CastRay(Point3 origin, Vector3 direction){

    // Do some work
}

void main(){
    Vector3 origin      = new Vector3;
    Vector3 direction   = new Vector3;

    CastRay(origin,direction);
    CastRay(direction,origin);
}


-- INSERT --

mike:~$ dmd alias.d -of=al
```

Vec3 and Unit Test

- One of the huge wins (and this saved me many times) is unittest in DLang
    - I followed test driven development for the Vec3 library
    - This Vec3 **library has to work** so it is critical I have confidence it worked.

```d
/// Unit vector tests
unittest{
    Vec3 v1 = new Vec3(2,3,4);
    Vec3 v2 = new Vec3(1,0,0);

    assert(v1.IsUnitVector() == false);
    assert(v2.IsUnitVector() == true);
    assert(v1.ToUnitVector().IsUnitVector() == true);

    Vec3 v3 = new Vec3(0.5,0.25,0.115);
    assert(v3.ToUnitVector().IsUnitVector() == true);

    Vec3 v4 = new Vec3(0.0,0.0,0.0);
    assert(v4.ToUnitVector().IsUnitVector() == false);

    Vec3 v5 = new Vec3(1.96,2.98,3.1);
    assert(v5.ToUnitVector().IsUnitVector() == true);

    Vec3 v6 = new Vec3(-0.98,0.97,0.0);
    assert(v6.ToUnitVector().IsUnitVector() == true);
}
```

Vec3 and Unit Test - Bug Caught (1/2)

- More than one time I found a ‘divide by 0’ error that I was not checking for.
    - You can see the amendment on the right I made to check for this.

```
/// Retrieve a unit vector
Vec3 ToUnitVector(){
    // Compute the length once
    auto len = Length();
    if(len!=0){
        e[0] = e[0] / len;
        e[1] = e[1] / len;
        e[2] = e[2] / len;
    }
    return this;
}
```

Vec3 and Unit Test - Bug Caught (1/2)

- More than one time I found a ‘divide by 0’ error that I was not checking for.
  - You can see the amendment on the right I made to check for this.

```
/// Retrieve a unit vector
Vec3 ToUnitVector(){
    // Compute the length once
    auto len = Length();
    if(len!=0){
        e[0] = e[0] / len;
        e[1] = e[1] / len;
        e[2] = e[2] / len;
    }
    return this;
}
```

Vec3 and Unit Test - Bug Caught (2/2)

- Bug catching also led to more helper functions being created, and more unit tests being written.
    - (And evening finding more properties like ‘epsilon built in!)
    - Working in DLang is working in a software engineering language -- I really enjoy not having to fight the language or rely on third-party libraries for this.

```
/// Test if this is a zero vector
bool IsZero() const{
    return (abs(e[0]) <= 0.000001 &&
            abs(e[1]) <= 0.000001 &&
            abs(e[2]) <= 0.000001);
}
```

Properties for Floating Point Types

Property	Description
`.init`	initializer (NaN)
`.infinity`	infinity value
`.nan`	NaN value
`.dig`	number of decimal digits of precision
`.epsilon`	smallest increment to the value 1

Vec3 Math Class - Operator Overloading

- Operator overloading I found overall pleasant enough to implement.
    - A Vec3 class or a Matrix4x4 would be examples in graphics of classes worth implementing operator overloading.

```
/// Handle multiplication and division of a scalar
/// for a vector
Vec3 opBinary(string op)(double rhs){
    Vec3 result = new Vec3(0.0,0.0,0.0);
    if(op=="*"){
        result[0] = e[0] * rhs;
        result[1] = e[1] * rhs;
        result[2] = e[2] * rhs;
    }
    else if(op=="/"){
        result[0] = e[0] / rhs;
        result[1] = e[1] / rhs;
        result[2] = e[2] / rhs;
    }
    else if(op=="+"){
        result[0] = e[0] + rhs;
        result[1] = e[1] + rhs;
        result[2] = e[2] + rhs;
    }
    else if(op=="-"){
        result[0] = e[0] - rhs;
        result[1] = e[1] - rhs;
        result[2] = e[2] - rhs;
    }
    return result;
}
```

Vec3 Math Class - Leveraging Uniform Function Call Syntax (UFCS)

- The last thing I'll say about Vec3, is that very often I am going to want to perform multiple operations on a Vec3
    - (e.g. Getting the direction of the vector and then normalizing it)
- Uniform Function Call Syntax (UFCS) [link to tour] makes this beautiful, and just helps with DLang being a very clean language

```
(r.GetDirection().ToUnitVector()
```

Vec3 And Drawing Spheres

- Finally, once Vec3 is properly implemented, I can draw a sphere.
  - Note:
    - The blueish colors are from how I'm coloring the sphere using its normals (imagine the normals as sticking out like a porcupine -- perpendicular to each face of the sphere, we encode the direction of the vector normal into the color of the sphere.)

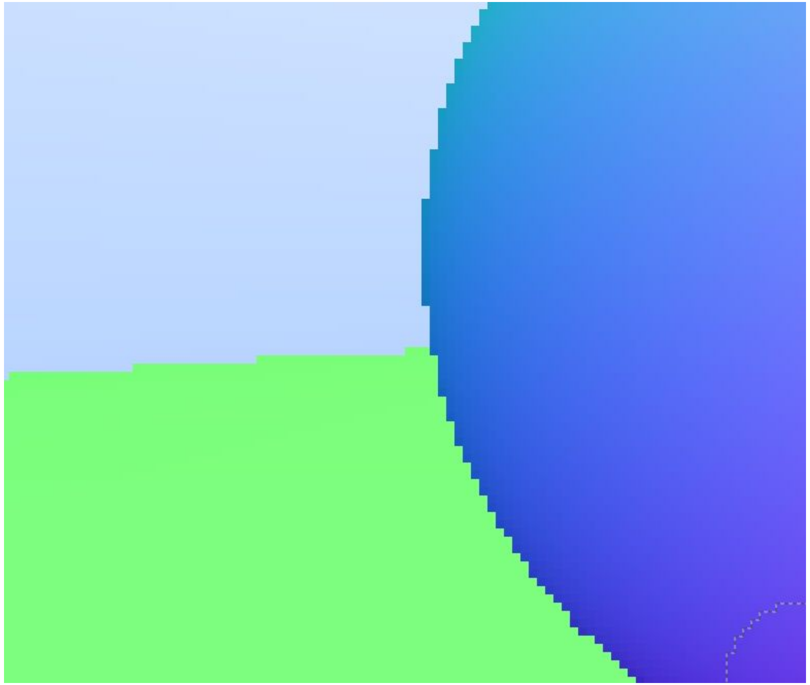
Something is not quite right--Maybe you'll notice if you have 20/20 vision

Drawing Better Spheres

Hours ~11-13

Anti-aliasing

- In graphics we can end up with ‘noisy’ edges.
  - Observe very closely the ‘jaggies’ and how we don’t see a perfectly smooth sphere.
- To fix this, we essentially ‘**sample randomly**’ neighboring pixels
  - (In practice, this means shooting multiple rays per pixel to accumulate the final color)



DLang has std.random to help!

- I am able to leverage std.random in this case [documentation link]
  - A variety of random functions and distributions exist that are built-in, which is wonderful!

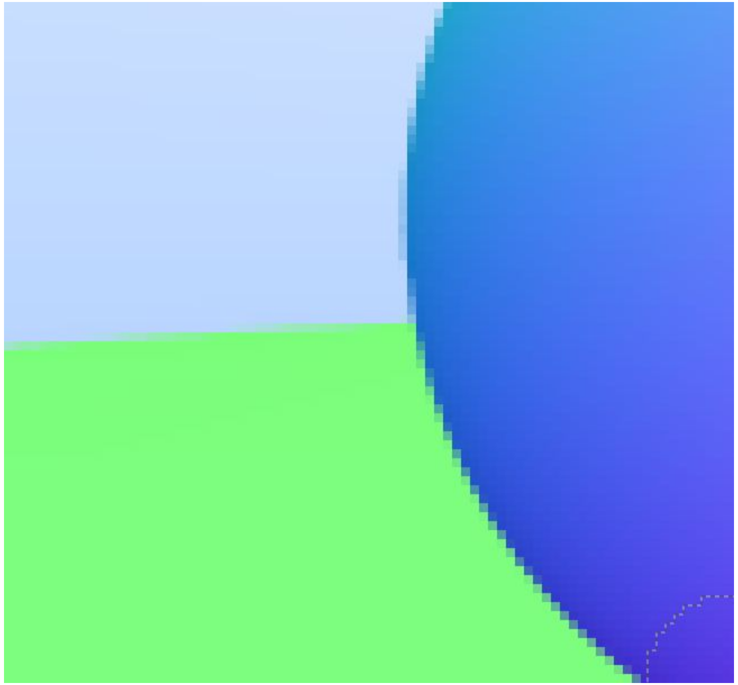
```d
module utility;

import std.random;

/// Generate a random double from 0..1
double GenerateRandomDouble(){
    auto rnd = Random(unpredictableSeed);
    return uniform01(rnd);
}


/// Generate a random double from a range
double GenerateRandomDouble(double min, double max){
    auto rnd = Random(unpredictableSeed);
    return min + (max-min)*uniform01(rnd);
}


/// Test range of GenerateRandomDouble()
unittest{
    assert(GenerateRandomDouble() < 1.00001);
    assert(GenerateRandomDouble() > -0.000001);
}

/// Test range of GenerateRandomDouble()
unittest{
    assert(GenerateRandomDouble(5,9) > 4.999999);
    assert(GenerateRandomDouble(5,9) < 9.000001);
}
```

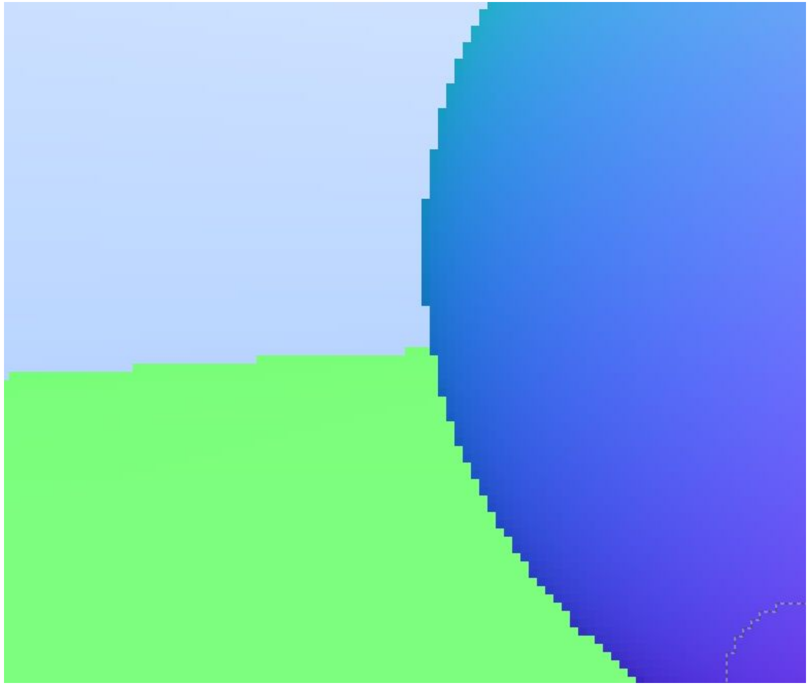
DLang has std.random to help!

- Note on my code on the right, DLang supports operator overloading,
  - I can write a few helpful functions to generate a single random value, or value between a range.
  - (Operator overloading not available in C for example)

```d
module utility;

import std.random;

/// Generate a random double from 0..1
double GenerateRandomDouble(){
    auto rnd = Random(unpredictableSeed);
    return uniform01(rnd);
}


/// Generate a random double from a range
double GenerateRandomDouble(double min, double max){
    auto rnd = Random(unpredictableSeed);
    return min + (max-min)*uniform01(rnd);
}


/// Test range of GenerateRandomDouble()
unittest{
    assert(GenerateRandomDouble() < 1.00001);
    assert(GenerateRandomDouble() > -0.000001);
}

/// Test range of GenerateRandomDouble()
unittest{
    assert(GenerateRandomDouble(5,9) > 4.999999);
    assert(GenerateRandomDouble(5,9) < 9.000001);
}
```

Left (Improved using antialiasing) | Right (hard edges)

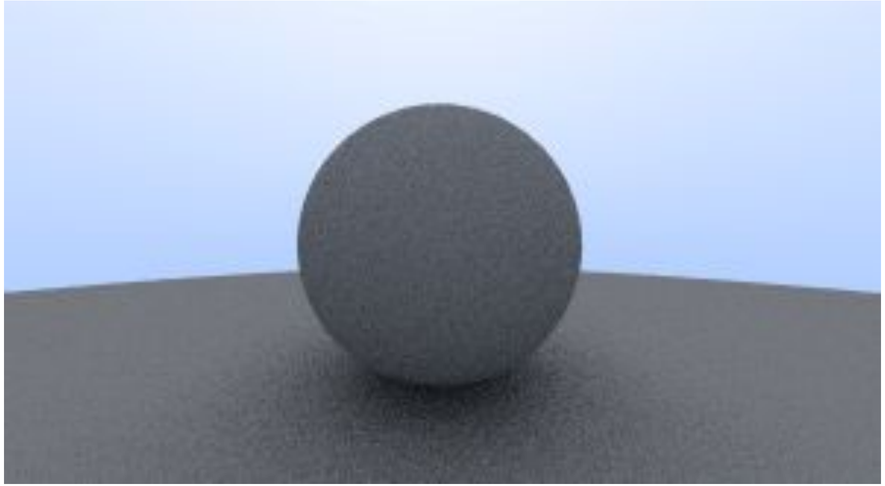




Handling Multiple Objects

Hours 14-18 (very roughly, maybe?)

One small sphere on top of a really big sphere

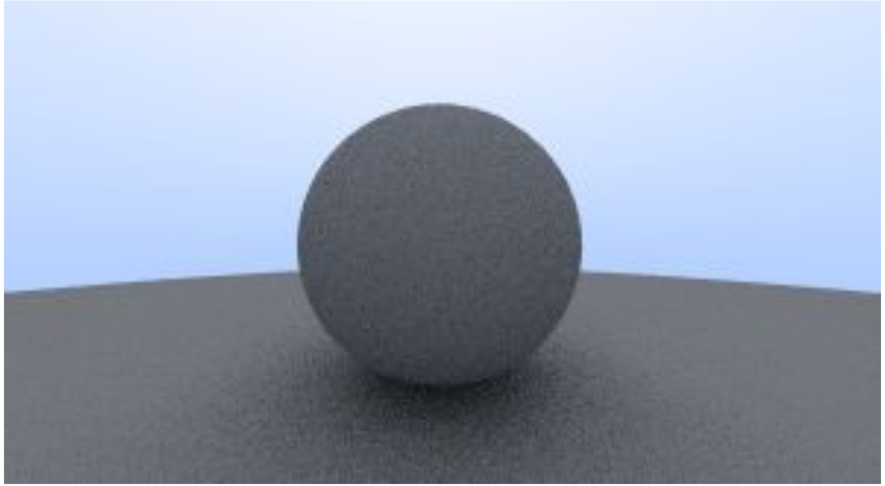


Chapter8

Talks / 2022_dconf_London

Rendering and Managing Multiple Shapes (1/2)

- We’re now ready to start rendering multiple spheres now that we’re confident one sphere will work.
- The next step is to build a ‘container’ to hold our objects.
  - Or do we need to do anything?
- D’s built-in data structures give us exactly what we want--no extra work!
  - `Hittable[]` is a dynamic array that we can append new objects to at run-time.
  - (Otherwise, many other containers in https://dlang.org/phobos/ are available).



```
/// Member variables
private Hittable[] m_objects;
```

Rendering and Managing Multiple Shapes (2/2)

- We'll also have to think a little bit about if we want to handle other shapes
  - i.e. not just spheres.
- Dlang supports [interfaces](), which allow us to derive a class from common interface, where we must implement the member functions of the interface.

```
interface Hittable{
    bool Hit(Ray r, double tMin, double tMax, ref HitRecord rec);
}
```

```
class Sphere : Hittable{

    /* ... */

    /// Test for intersection with a sphere
    bool Hit(Ray r, double tMin, double tMax, ref HitRecord rec){
```

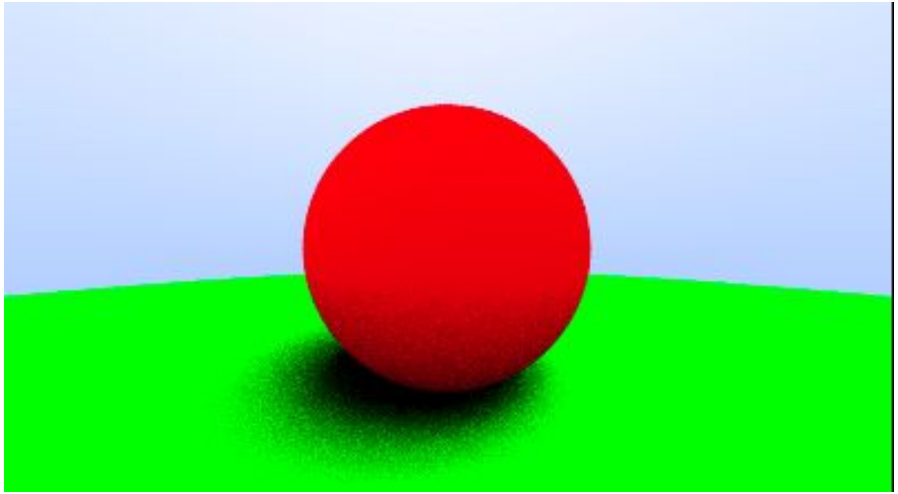
```
interface Hittable{
    Hit(...);
}
```

```
class Sphere : Hittable{
    /* ... */
}
```

Grand Finale and Final Notes

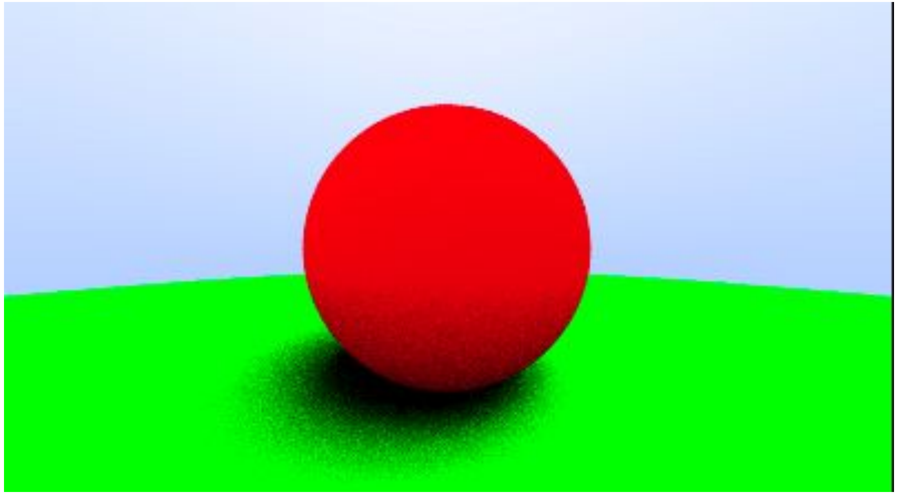
Hours 19-24 (To be honest, I lost count, but it was close to 2 full work day sittings)

Utilizing different materials



Scaling

- Our goal is to start getting something more interesting
  - More objects
  - More colors
  - Maybe even different materials!



Multiple Materials

- I will use the same strategy as I did for handling multiple objects
  - 1 common interface that each derived class must implement.
  - This will allow me to keep a relatively flat inheritance hierarchy, and also make sure I implement all needed member functions.

```
interface Material{
    scatter();
}
```

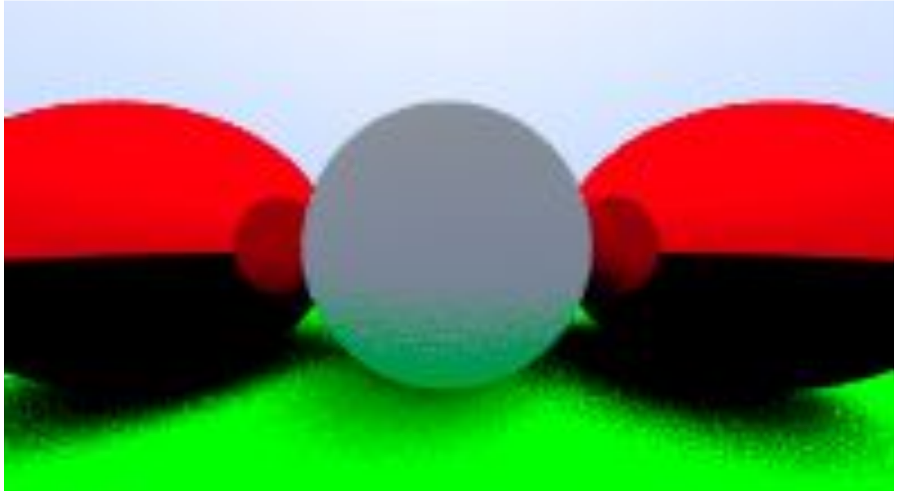
```
class Metal : Material{
    /* ... */
}
```

```
class Color : Material{
    /* ... */
}
```

A Few Software Engineering Things

Approaching the end
The next weekend...

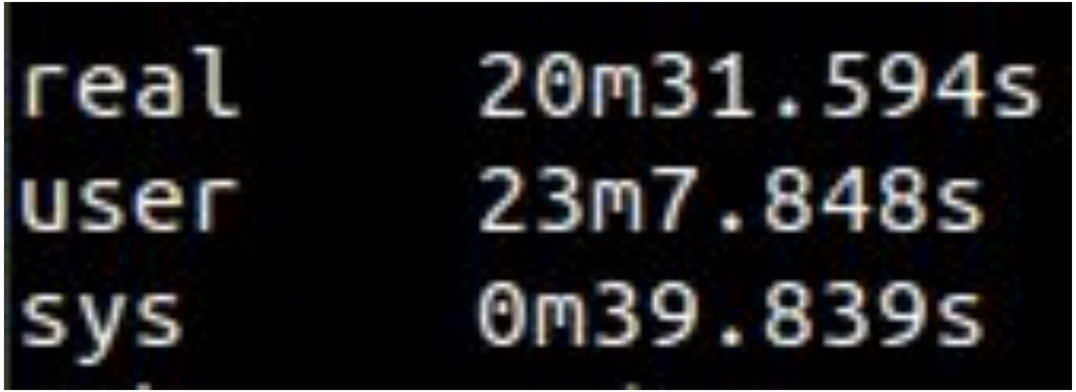
The final product!

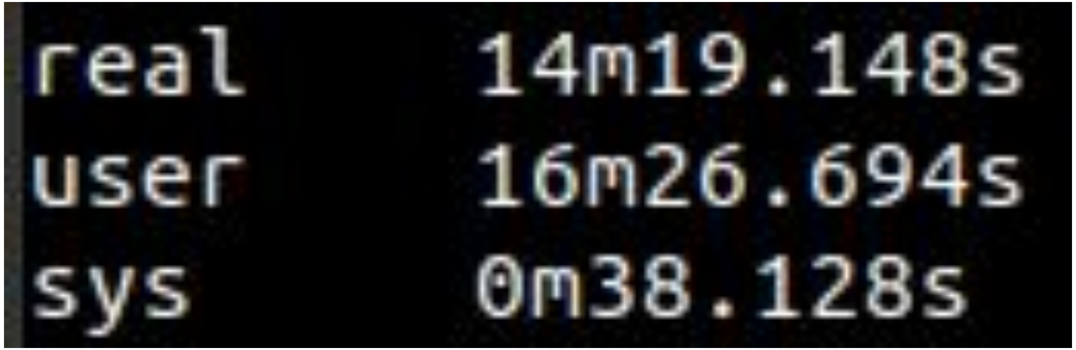


A Few Software Engineering Things

Performance

- Debug Build of Ray Tracer (Top)
  - Ouch!
    - 20 minutes and 31 seconds!
  - Did I mention ray tracing can be expensive?
- Release Build (Bottom) of Ray Tracer a bit faster at 14 minutes 19 seconds





Performance - Parallelism

- Good news, DLang has parallelism built-in.
    - (And SIMD Vector Extensions)
- Ray Tracing is something that is massively parallel, so this is worth reporting on at a later date, maybe a future talk?
    - Also, I will do more with -profile=gc in the future
- (Pssst, the slow speed is also a major data structure problem-- bounding volume hierarchy will also improve sparse scenes significantly)

std.parallelism

The module `std.parallelism` implements high level primitives for convenient concurrent programming.

parallel

`std.parallelism.parallel` allows to automatically distribute a `foreach`'s body to different threads:

```
// parallel squaring of arr
auto arr = iota(1,100).array;
foreach(ref i; parallel(arr)) {
    i = i*i;
}
```

https://tour.dlang.org/tour/en/multithreading/std-parallelism

Debugging (1/2)

- Spot the bug!

```
            // World
            HittableList world;
            world.Add(new Sphere(new Vec3(0,0,-1),0.5));
            world.Add(new Sphere(new Vec3(0,-100.5,-1),100));
```

Debugging (2/2)

- Spot the bug!
- D works wonderfully with GDB on my linux machine for catching these things.
    - (This was using the DMD compiler today. Perhaps GCC integratess even better?)

```
// World
HittableList world;
world.Add(new Sphere(new Vec3(0,0,-1),0.5));
world.Add(new Sphere(new Vec3(0,-100.5,-1),100));
```

```
// World
HittableList world = new HittableList;
world.Add(new Sphere(new Vec3(0,0,-1),0.5));
world.Add(new Sphere(new Vec3(0,-100.5,-1),100));
```

And More!

- I did not make heavy use of code coverage, but it is another nice feature in dmd
  - https://dlang.org/articles/code_coverage.html
- My build system was slowly hacked together into two scripts to run unit tests and then the build.
  - sh test.sh && sh ./build.sh
  - In reality, I will use dub if I introduce any additional dependencies.
    - Having a package manager is a big deal, and a very good thing in my opinion!
- Overall, it is clear to me that years of software experience have gone into DLang, and it makes it a very fun language to build real software with!

Resources for More on Ray Tracing

- Ray Tracing
  - Ray Tracing in One Weekend Series (**There are 3 free books)**
    - **https://raytracing.github.io/**
  - Disney's Practical Guide to Path Tracing (Video: 9 minutes)
  - Ray Tracing Essentials [Part 1 - Part 7]
    - https://developer.nvidia.com/blog/ray-tracing-essentials-part-1-basics-of-ray-tracing/
  - Ray Tracing Course from SIGGRAPH
    - (class 1 link) https://www.youtube.com/watch?v=3xMeKal2-Ws
- Dlang
  - Ali Çehreli
    - Programming in D (free online and paperback book)
  - DLang Tour
    - https://tour.dlang.org/

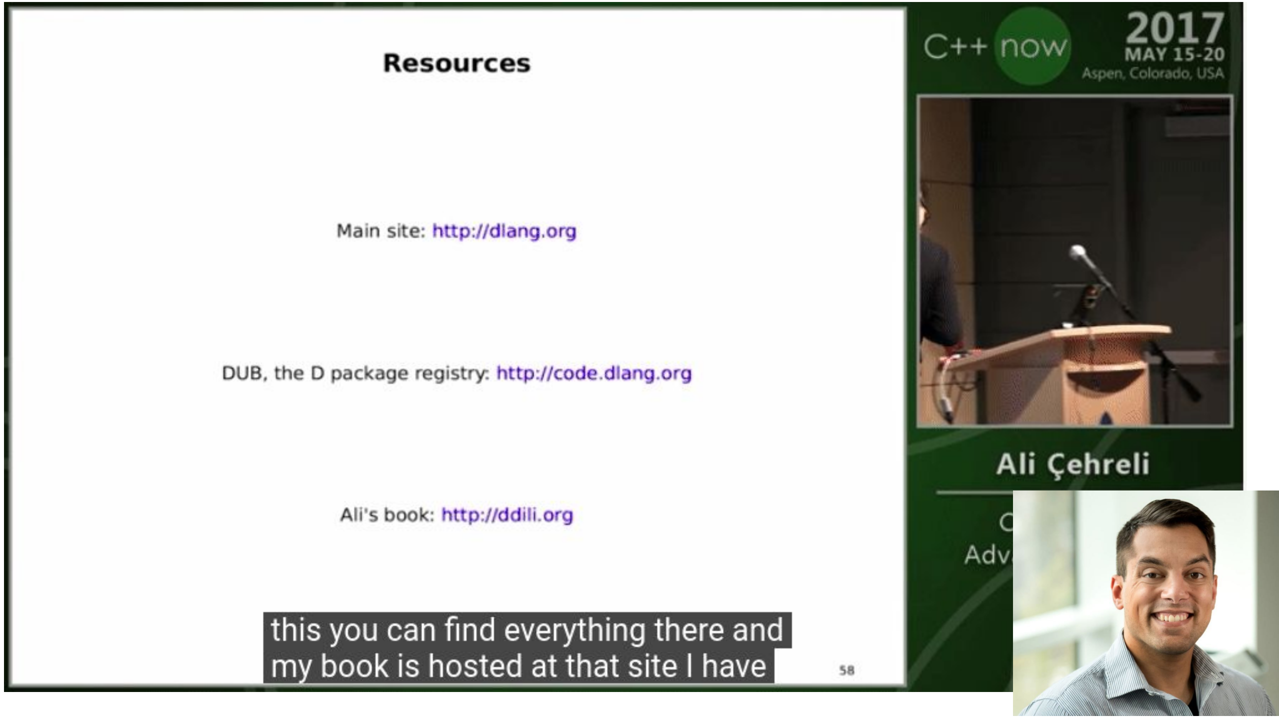
Resources for More on Ray Tracing

- Ray Tracing
  - Ray Tracing in One Weekend Series (**There are 3 free books)**
    - **https://raytracing.github.io/**
  - Disney's Practical Guide to Path Tracing (Video: 9 minutes)
  - Ray Tracing Essentials [Part 1 - Part 7]
    - https://developer.nvidia.com/blog/ray-tracing-essentials-part-1-basics-of-ray-tracing/
  - Ray Tracing Course from SIGGRAPH
    - (class 1 link) https://www.youtube.com/watch?v=3xMeKal2-Ws
- Dlang
  - **Ali Çehreli**
    - **Programming in D (free online and paperback book)**
  - DLang Tour
    - https://tour.dlang.org/

This one in particular is a great resource!

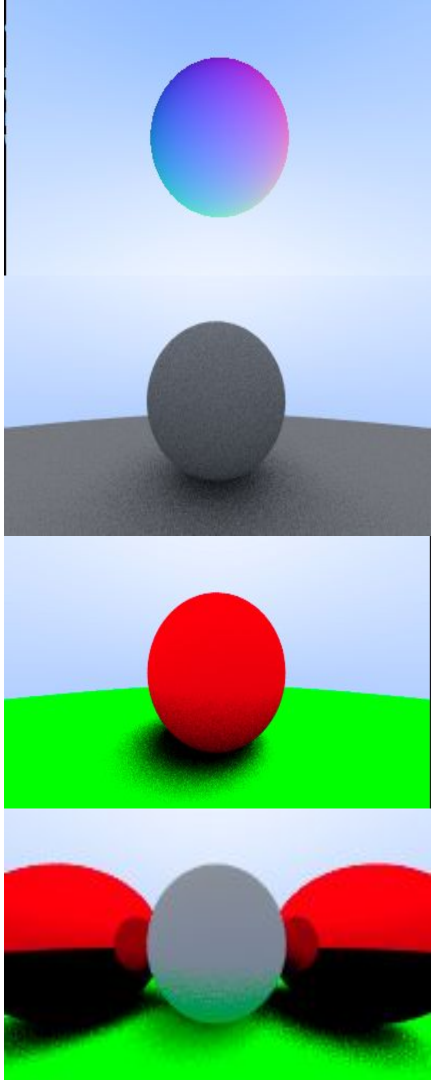
Thanks Ali!

- I'm not sure Ali is going to remember gifting me a copy of his book at his "Competitive Advantage with D" Meeting C++ 2017 talk
  - https://youtu.be/vYEKEIpM2zo?t=1614 (see the moment!)
- But here you are, and thank you for being generous!s!



Summary

- In summary, I hope you have enjoyed this journey learning or refreshing on why DLang is a wonderful language to work in.
    - Building a Ray Tracer is an excellent project to exercise your skills on in the language.
- Your homework is to now go build a Ray Tracer and then post on Twitter :)
    - Tag @Peter_shirley
    - Use the #dlang
    - (And optionally tag me or this talk if it helped inspire you)



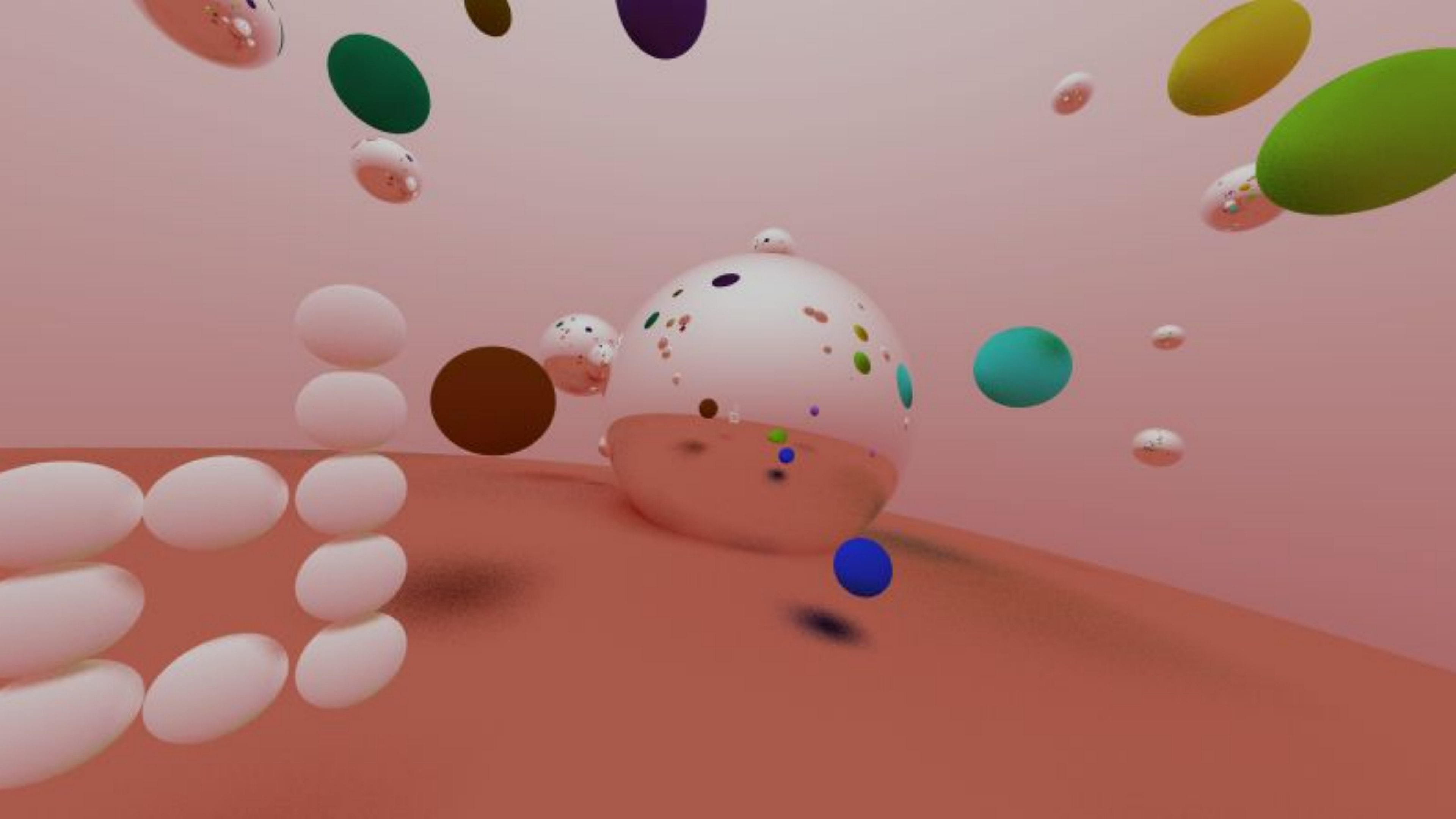
Bonus Content

DLang - YouTube Playlist

- For fun announcement
  - I’ve started a brand new series on YouTube on learning DLang.
  - This will be a long running series on learning the DLang.
- https://www.youtube.com/watch?v=HS7X9ERdjM4&list=PLvv0ScY6vfd9Fso-3cB4CGnSlW0E4btJV&index=1
  - (Series starts this August, maybe after this talk is broadcast again)

One more image ...

(Do you see the hidden easter egg?)



DCONF '22 LONDON

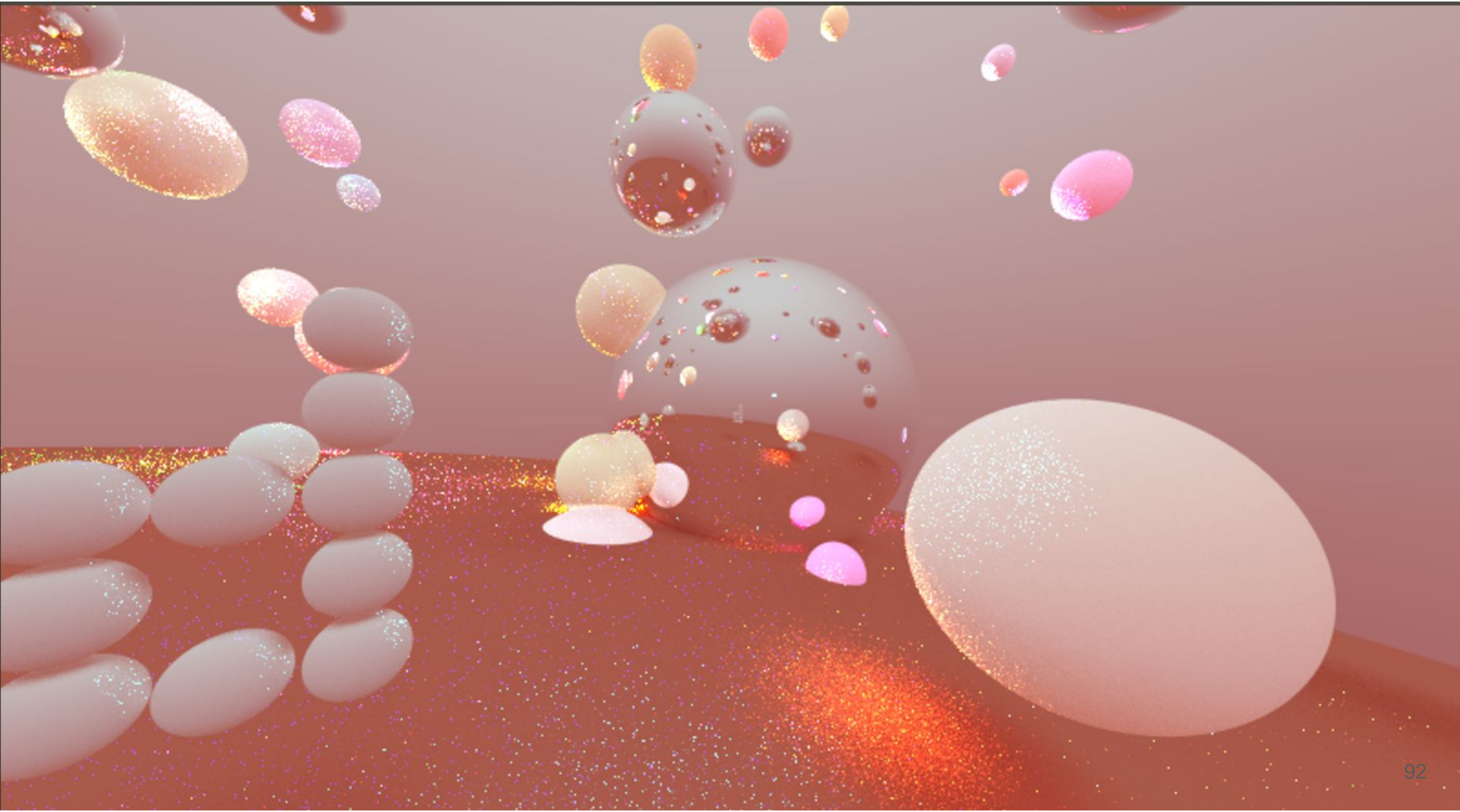
Thank you!

Ray Tracing in (Less than) One Weekend with Dlang

Social: @MichaelShah
Web: mshah.io
Courses: courses.mshah.io
YouTube: www.youtube.com/c/MikeShah

Presentor: Mike Shah, Ph.D.
13:30-14:15, Tue, August, 2 2022
45 minutes | Introductory Audience

Bloopers



Unused

Outline

• Brief Introduction to what Ray Tracers are and Peter Shirley's Book

◦ Will show my C++ and Python Ray Tracers previously implemented

• Will through creating an image class

◦ Highlight easy file operations and string parsing in Dlang versus C++

• Walk through building a vec3 class

◦ Highlighting operator overloading in Dlang (contrasting also with Python)

◦ Will also show how integrating unit tests within vec3 class built confidence in code.

• Walk through creating a sphere and background.

◦ Will highlight a few usages of std.algorithm

• Walk through a few more advanced features (providing an overview) of Ray tracers:

Antialiasing, and materials

• Walk through adding a camera class and scene class to organize the project

◦ Will show how I used 'ddoc' to document as I wrote code to document these features.

• Will show an example scene at the end

◦ But one more trick, will show how using std.concurrency (and/or) std.parallelism I could trivially improve performance

• Conclusion and final thoughts

◦ Why I'm considering teaching software engineering courses in Dlang based on the features presented.

Useful resources

- https://wiki.dlang.org/Programming_in_D_for_C%2B%2B_Programmers
- https://dlang.org/articles/cpptod.html
- https://p0nce.github.io/d-idioms/#How-does-D-improve-on-C++17?
- D Style Guide
  - https://dlang.org/dstyle.html

Vec3 Math Class - memoize

- Maybe talk about how I tried to memoize vec3 and then profile the performance.

Maybe squeeze in ‘static if’ for flipping an image