

INFOGR – Computer Graphics

Jacco Bikker & Debabrata Panja - April-July 2017

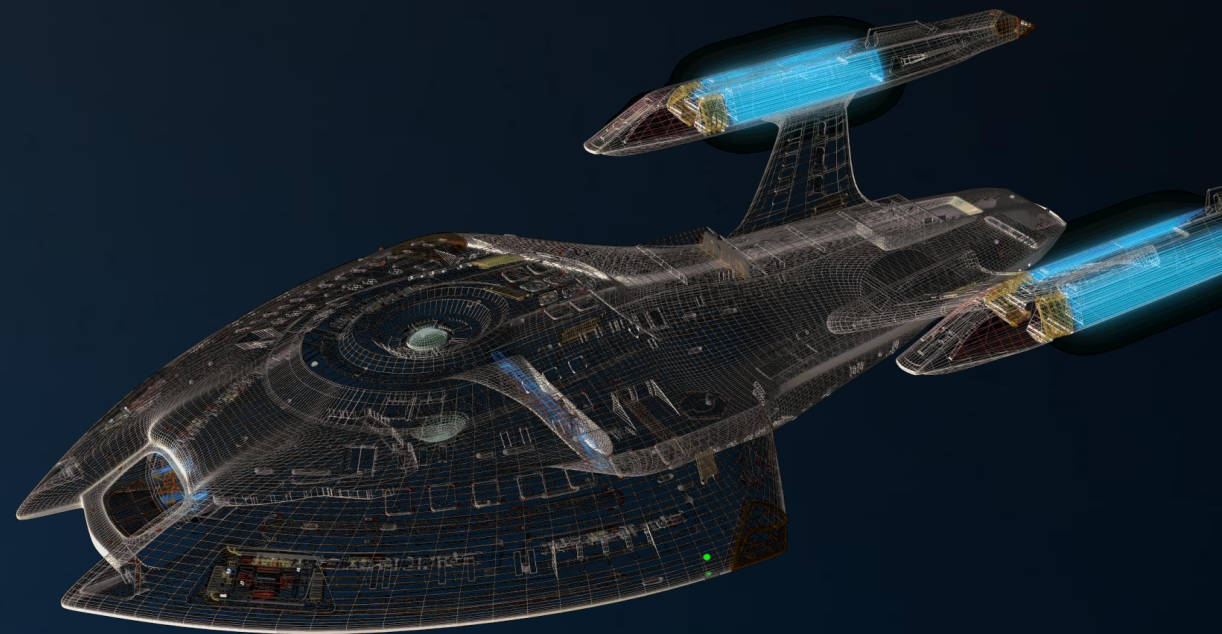
Lecture 1: “Introduction”

Welcome!



Today's Agenda:

- Graphics
- Course Introduction
- Field Study
- Rasters
- Colors



Introduction



Just Cause 3



Introduction



HALO 5: GUARDIANS MULTIPLAYER BETA

HALO 5



Introduction



GTA V



Introduction



Homeworld Remastered



Introduction

```

ics
& (depth < MAXDEPTH)
{
    // Inside / Outside
    int nt = nt / nc;
    float r2t = 1.0f - nnt;
    float r2b = 1.0f - nnt;
    float r2g = 1.0f - nnt;
    float r2r = 1.0f - nnt;
    float r2 = (r2t + r2b + r2g + r2r) / 4.0f;
    float R = (D * nnt + N * (1.0f - nnt));
    float E = diffuse;
    bool refl = true;
    // Refractive
    float refl + refr) && (depth < MAXDEPTH)
    {
        // Inside / Outside
        float D, N;
        float refl * E * diffuse;
        bool refl = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    // Estimation - doing it properly, closely
    if;
    radiance = SampleLight( &rand, I, &t, &light );
    float x = radiance.x + radiance.y + radiance.z;
    if (x > 0) && (x < 1)
    {
        w = true;
        float brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
        float t3 factor = diffuse * INVPI;
        float weight = Mis2( directPdf, brdfPdf );
        float cosThetaOut = dot( N, L );
        float E = ((weight * cosThetaOut) / directPdf) * (radiance.x + radiance.y + radiance.z);
    }
    // Random walk - done properly, closely following survival
    survive);
    // SampleDiffuse
    float brdf = SampleDiffuse( diffuse, N, r1, r2, &R );
    float survive;
    float pdf;
    float n = E * brdf * (dot( N, R ) / pdf);
    bool refl = true;

```

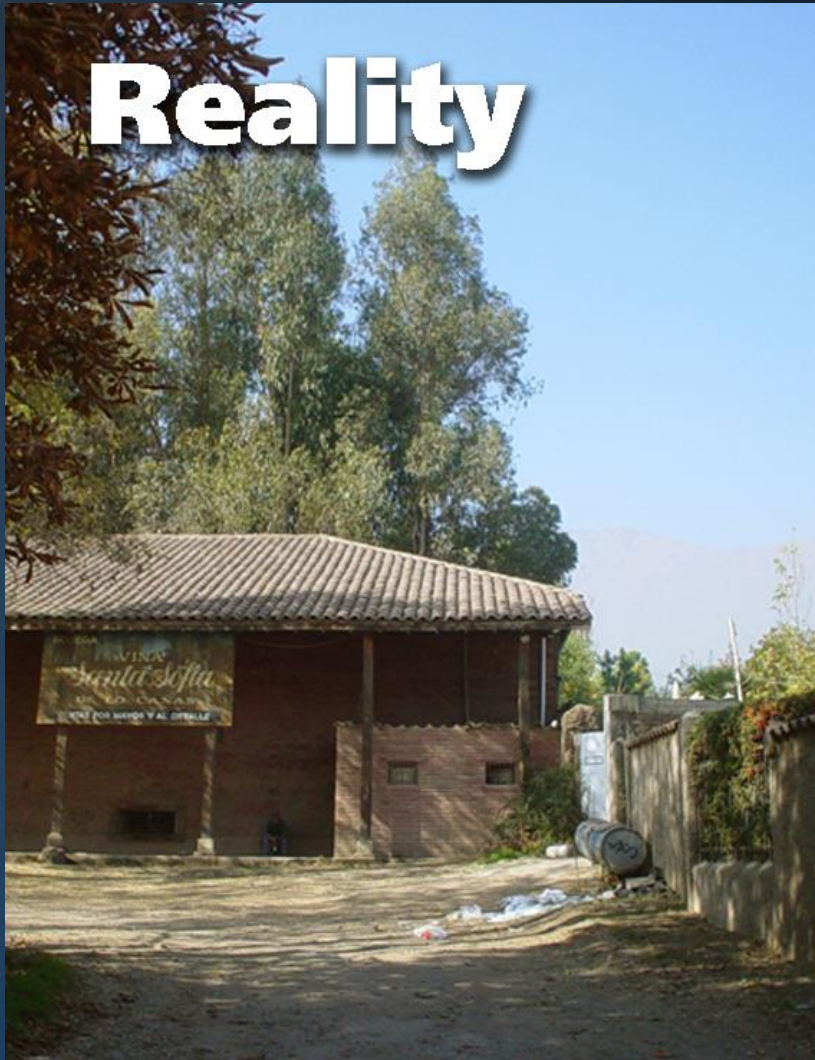


Crysis

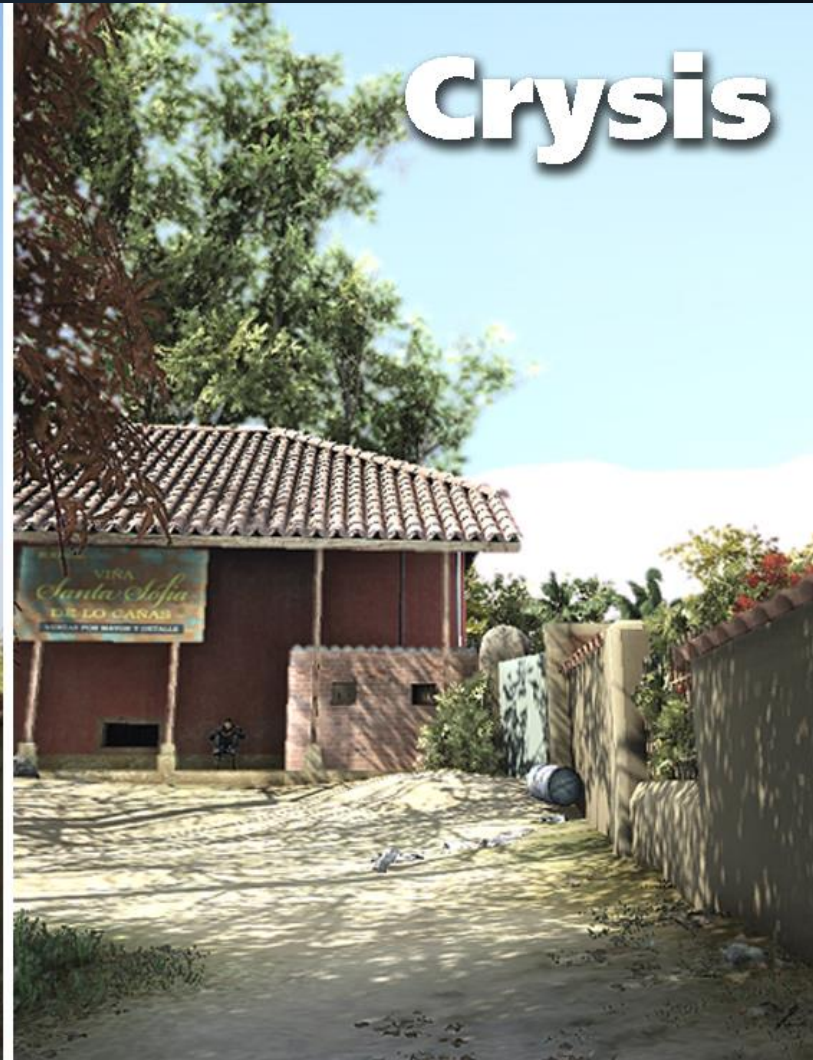


Introduction

Reality



Crysis



Crysis



Introduction

```
...ics
& (depth < MAXDEPTH)
{
    nt = inside / 1.5;
    nt = nt / nc;
    nnt = nt * nt;
    nnt2t = 1.0f - nnt;
    D, N );
}

at a = nt - nc, b = nt - nc;
at Tr = 1 - (R0 + (1 - R0) * nnt);
Tr) R = (D * nnt - N * (D0 - R0));

E * diffuse;
= true;

efl + refr)) && (depth < MAXDEPTH)
{
    D, N );
    refl * E * diffuse;
    = true;

MAXDEPTH)

survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &t, &light );
e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
{
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
    random walk - done properly, closely following
    survive)

    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
}
```



Lord of the Rings



Introduction

```
...ics
& (depth < MAXDEPTH)
{
    // Inside / Outside
    nt = inside / (1 + inside);
    nc = nt / nc; ddc = ddc / nc;
    cos2t = 1.0f - nnt; // nnt = nt * nt
    D, N );
}

// Russian roulette
// at a = nt - nc, b = nt * nc;
// at Tr = 1 - (RB + (1 - RB) * a);
// Tr) R = (D * nnt - N * (ddc
//
// E * diffuse;
// = true;
//
//
// refl + refr)) && (depth < MAXDEPTH)
//
// D, N );
// refl * E * diffuse;
// = true;
//
// MAXDEPTH)
//
// survive = SurvivalProbability( diffuse );
// estimation - doing it properly, closely follow
// if;
// radiance = SampleLight( &rand, I, &t, &align
// e.x + radiance.y + radiance.z) > 0) && (survive)
//
// w = true;
// at brdfPdf = EvaluateDiffuse( L, N ) * Pdiff;
// at3 factor = diffuse * INVPI;
// at weight = Mis2( directPdf, brdfPdf );
// at cosThetaOut = dot( N, L );
// E * ((weight * cosThetaOut) / directPdf) *
//
// random walk - done properly, closely following
// survive)
//
// at3 brdf = SampleDiffuse( diffuse, N, r1, r2, r3 );
// survive;
// pdf;
// n = E * brdf * (dot( N, R ) / pdf);
// sion = true;
```



Furious 7 (Paul Walker)



Introduction

```
...ics
& (depth < MAXDEPTH)
{
    // Inside / Outside
    nt = nt / nc; rddo = rddo / nc;
    pos2t = 1.0f - nnt * nnt;
    D, N );
}

// Inside / Outside
at a = nt - nc, b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * pos2t);
Tr) R = (D * nnt - N * pos2t);

E * diffuse;
= true;

// Inside / Outside
efl + refr)) && (depth < MAXDEPTH)
{
    D, N );
    refl * E * diffuse;
    = true;

MAXDEPTH)

survive = SurvivalProbability( diff
estimation - doing it properly, c
if;
radiance = SampleLight( &rand, 1, &
e.x + radiance.y + radiance.z > 0);

w = true;
at brdfPdf = EvaluateDiffuse( L, N );
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / direc

random walk - done properly, closely
vive)

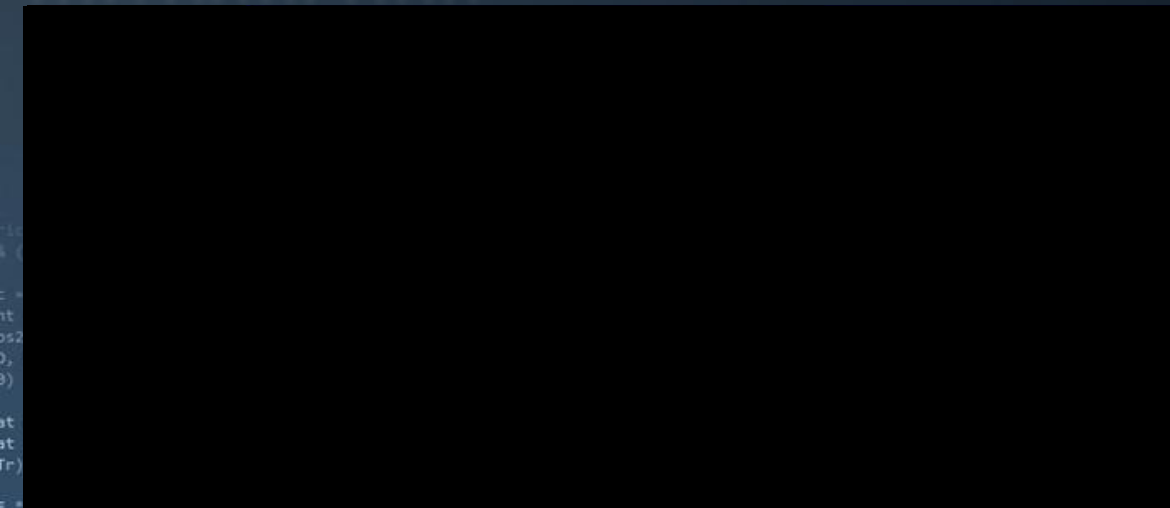
;
at3 brdf = SampleDiffuse( diffuse,
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```



Star Wars



Introduction



```

= true;
...
efl + refr)) && (depth < MAXDEPTH)
...
D, N );
refl * E * diffuse;
= true;
...
MAXDEPTH)
survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &t, &light );
e.x + radiance.y + radiance.z) > 0) && (cosThetaOut > 0)
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
at3 factor = diffuse * INVPI;
at weight = Mix2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following well
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &P
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
ision = true;

```

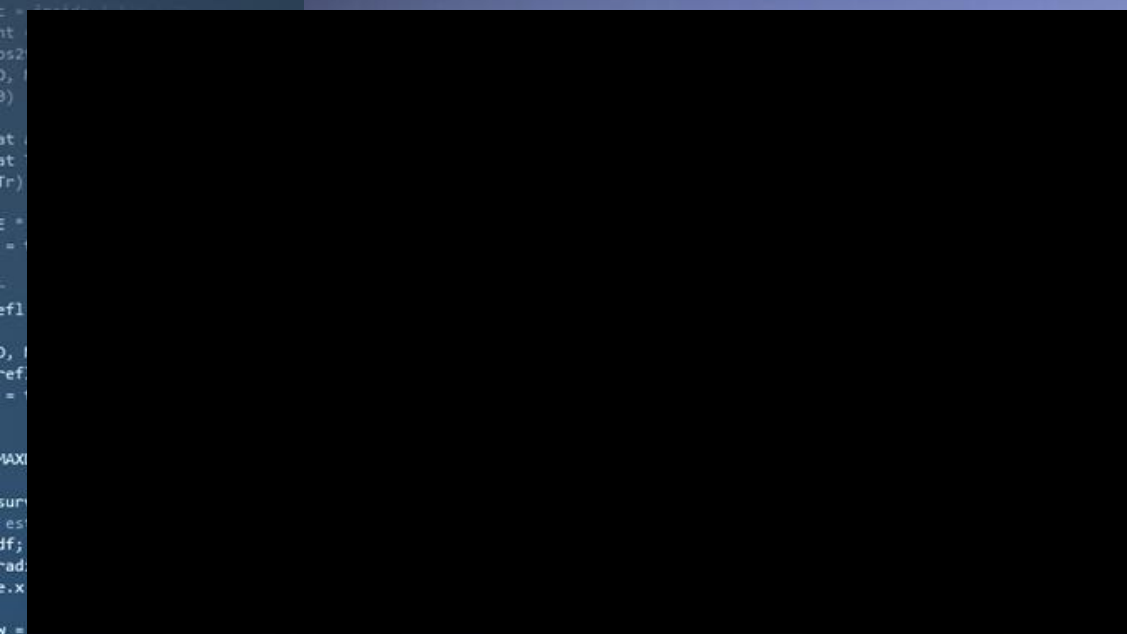


Zootopia



Introduction

```
...ics  
& (depth < MAXD);
```



```
...at brdfPdf = EvaluateDiffuse( L,  
at3 factor = diffuse * INVPI;  
at weight = Mis2( directPdf, brdf  
at cosThetaOut = dot( N, L );  
E * ((weight * cosThetaOut) / di
```

```
...andom walk - done properly, close  
...ive)
```

```
...at3 brdf = SampleDiffuse( diffuse, N, r1, r2, BR, xpdf);  
...urvive;
```

```
...pdf;  
...n = E * brdf * (dot( N, R ) / pdf);  
...ision = true;
```



Introduction



Just Cause 3

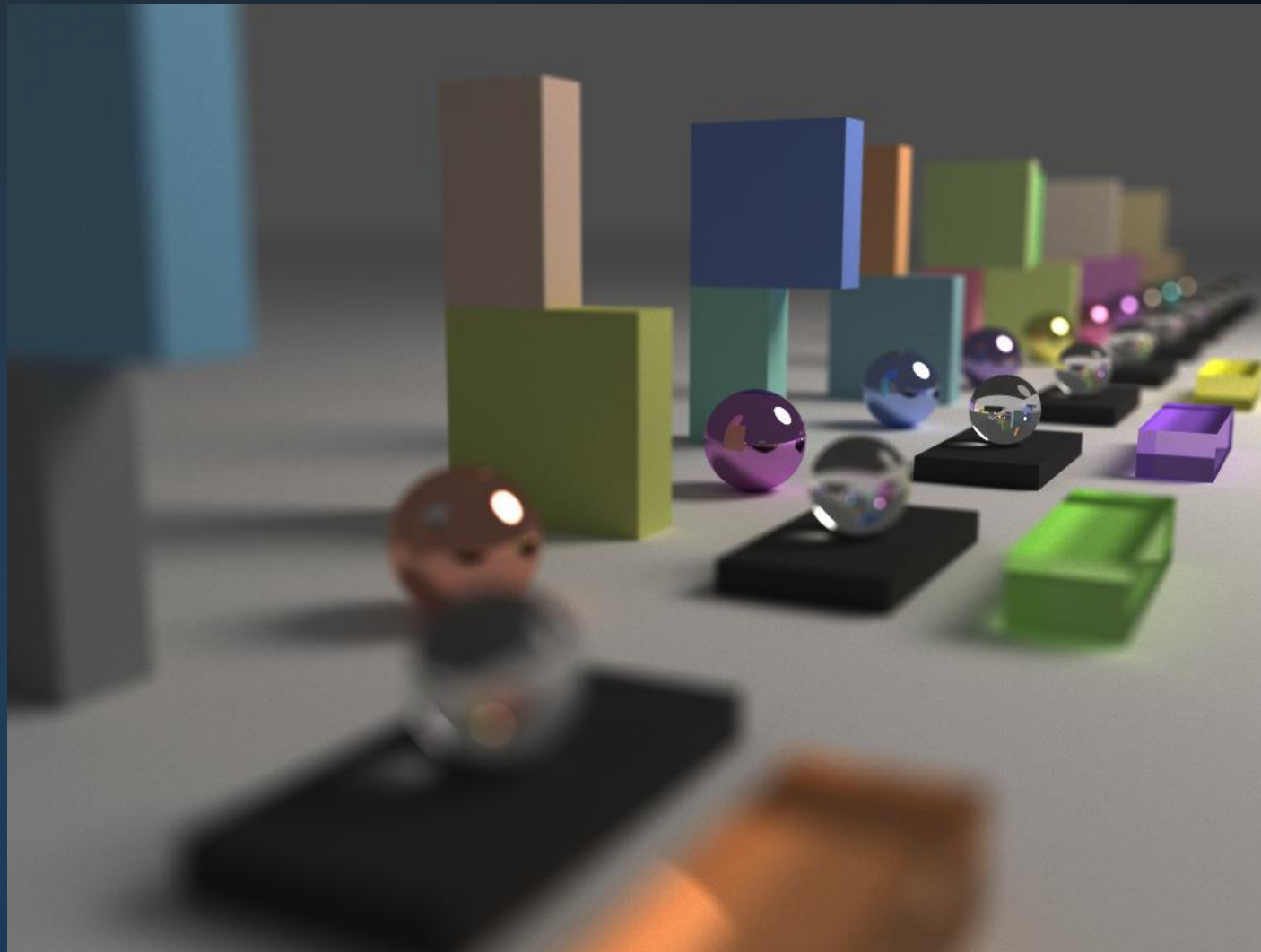


Introduction

```

    if (depth < MAXDEPTH)
    {
        // Inside the sphere
        Vec r = inside / 1.5;
        Vec nt = nt / nc, nde = nde / nc;
        double cos2t = 1.0f - nnt * nnt;
        if (cos2t < 0) cos2t = 0;
        Vec t(D, N);
        Vec a = nt - nc, b = nt + nc;
        double r0 = 1 - (R0 + (1 - R0) * cos(2 * PI * rand()));
        double R = (D * nnt - N * (nde * cos2t + 1));
        Vec E * diffuse;
        bool refl = true;
        Vec refl + refr)) && (depth < MAXDEPTH)
    {
        Vec D, N);
        Vec refl * E * diffuse;
        bool refl = true;
        Vec refl + refr)) && (depth < MAXDEPTH)
    {
        Vec survive = SurvivalProbability( diffuse );
        // estimation - doing it properly, closely following walk
        if (survive)
        {
            Vec radiance = SampleLight( &rand, I, &L, &align );
            Vec e.x + radiance.y + radiance.z > 0) && (depth < MAXDEPTH)
        {
            Vec w = true;
            Vec brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
            Vec t3 factor = diffuse * INVPI;
            Vec weight = Mis2( directPdf, brdfPdf );
            Vec cosThetaOut = dot( N, L );
            Vec E * ((weight * cosThetaOut) / directPdf) * (radiance);
            Vec random walk - done properly, closely following walk
            Vec survive);
        }
        Vec t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        Vec survive;
        Vec pdf;
        Vec n = E * brdf * (dot( N, R ) / pdf);
        Vec n = true;
    }

```

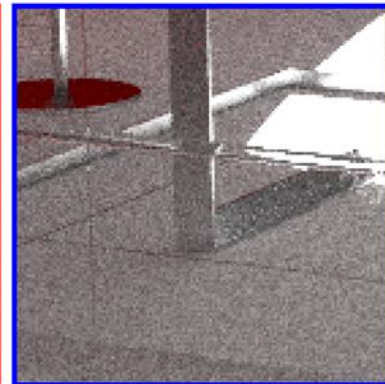
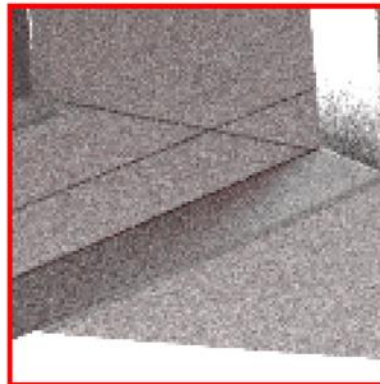
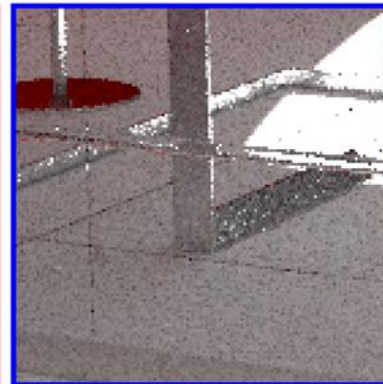
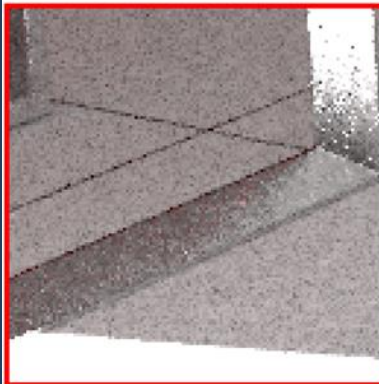




Introduction

```

    if (depth < MAXDEPTH)
    {
        // Inside the glass
        nt = inside / 1.5;
        nc = nt / (nc + nt);
        n2t2 = 1.0f - nc * nc;
        D = N * (nt > 0 ? 1 : -1);
        // Russian roulette
        float a = nt - nc, b = nt + nc;
        float Tr = 1 - (RB * (1 - RB));
        float R = (D * nnt - N * (a * Tr));
        // Diffuse
        E * diffuse;
        // Refractive
        refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        // Refractive
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely
    if;
    radiance = SampleLight( &rand, I, &L, &align,
    e.x + radiance.y + radiance.z) > 0) && (depth <
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    at3 factor = diffuse * INVPI;
    at weight = Mix2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (rad
    random walk - done properly, closely following the
    survive)
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R,
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
    
```



MCPPM, 180 min.

VCM, 180 min.

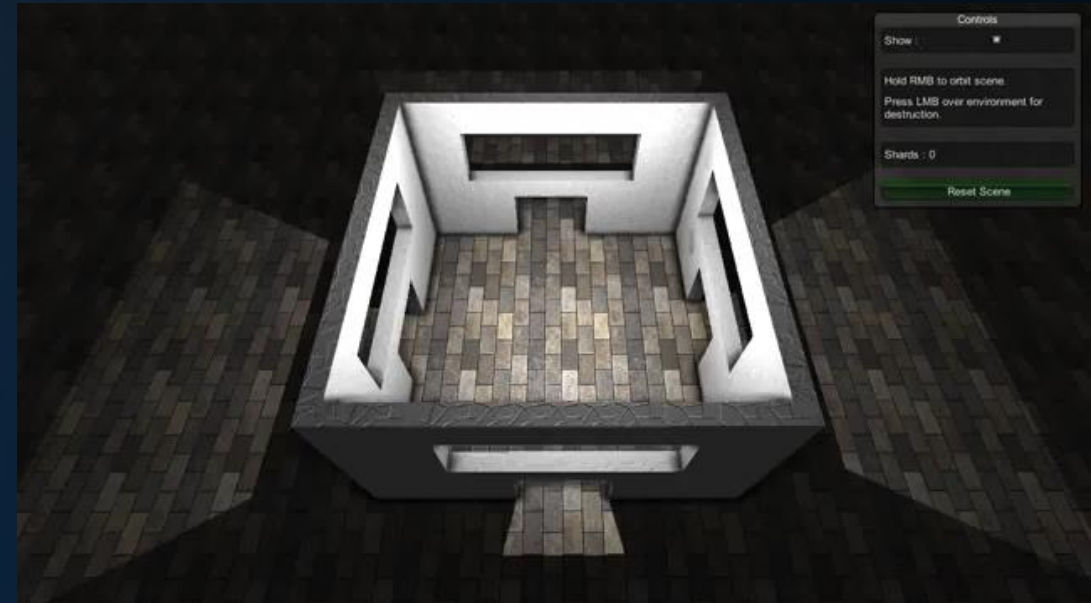
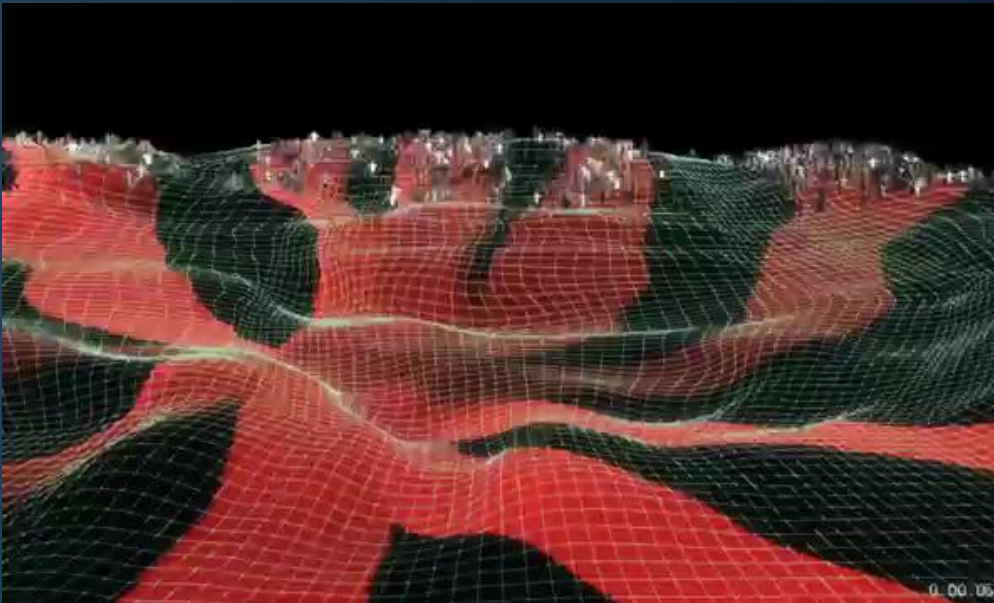


Introduction

```

    (depth < MAXDEPTH)
    {
        int inside = 1;
        int nt = nt / nc, ndn = ndn / nc;
        double cos2t = 1.0f - nnt / nnt;
        double N[3];
        double R[3];
        double a = nt - nc, b = nt + nc;
        double Tr = 1 - (R0 + (1 - R0) * cos2t);
        double R = (0 * nnt - N * (double)
            E * diffuse;
            = true;
            refl + refr)) && (depth < MAXDEPTH)
            0, N );
            E * diffuse;
            = true;
            refl + refr)) && (depth < MAXDEPTH)
            0, N );
            E * diffuse;
            = true;
            MAXDEPTH)
            survive = SurvivalProbability( diffuse );
            estimation - doing it properly, classically
            ff;
            radiance = SampleLight( &rand, I, &i, &light
            .x + radiance.y + radiance.z) > 0) && (max
            w = true;
            brdfPdf = EvaluateDiffuse( L, N ) * Psurf
            t3 factor = diffuse * INVPI;
            weight = Mis2( directPdf, brdfPdf );
            cosThetaOut = dot( N, L );
            E * ((weight * cosThetaOut) / directPdf) *
            random walk - done properly, closely following
            ve)
            ;
            t3 brdf = SampleDiffuse( diffuse, N, r1, r2
            survive;
            pdf;
            n = E * brdf * (dot( N, R ) / pdf);
            sion = true;
    }
}

```



Introduction

Computer Graphics 2017:

Looking for realism (in several wrong places):

1. Rasterization

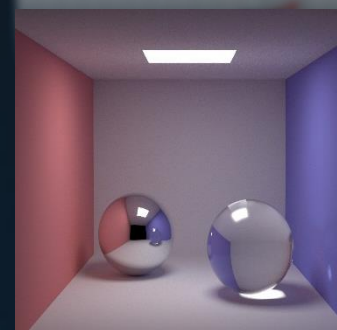
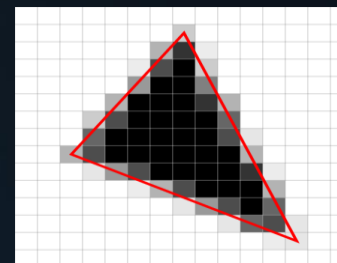
- Geometry
- Textures, shaders
- Clipping, culling
- Post processing
- ...

2. Ray tracing

- Ray/triangle intersections
- Bounding volume hierarchy
- Snell, Fresnel, Beer
- Whitted, Cook, Kajiya
- ...

3. Mathematics

- Vectors
- Matrices
- Transformations



Introduction

Language: English,
because of reasons.

Prerequisites: C#.

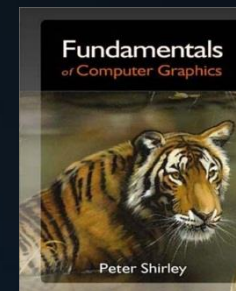
Literature: Fundamentals of Computer Graphics (3rd edition), by
Peter Shirley and Steve Marschner (or 4th, or 2nd, or 1st).

15 lectures.

Supporting math tutorials on Tuesdays (starting week 2).

Supporting working lectures on Thursdays (starting week 2).

For rooms: see schedule.



Introduction

Exams:

- Mid-term: May 23th.
- End of term: June 29th.
- Retake: July 13th.

Attendance:

You are not required to attend any of the lectures / tutorials / practicals (i.e., if you are here, it's because you want to).*

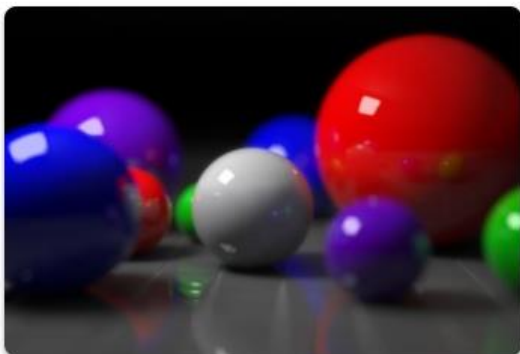
*Obviously, attendance is highly recommended.



Graphics

UNIVERSITEIT UTRECHT - INFORMATION AND COMPUTING SCIENCES

academic year 2016/17 – 4th period



Navigation

<http://www.cs.uu.nl/docs/vakken/gr>

Course Overview

Schedule

Practica

Literature & Links

News



Ctrl+1



Ctrl+2



Ctrl+3



Ctrl+4



infogr2017 ▾



● jbikker

All Threads

CHANNELS (2)



general

random

DIRECT MESSAGES



♥ slackbot

● jbikker (you)

+ Invite people

#general



1



0

Company-wide announcements and work-based matt...



Search



<https://infogr2017.slack.com/signup>

use uu.nl e-mail address

#general

You created this channel today. This is the very beginning of the [#general](#) channel. Purpose: *This channel is for team-wide communication and announcements. All team members are in this channel.* ([edit](#))

[+ Add an app or custom integration](#)[& Invite people to infogr2017](#)

Today

**jbikker** 9:55 AM

joined #general



Message #general



Introduction

Course characteristics:

This is a very intensive course. Be sure to keep up, i.e. don't miss lectures.

Be aware that this course will be attended by a diverse student population:

- Math-savvy students;
- Programming gurus;
- Game people;
- Informatics guys.

Regardless of your skill level and interests, make use of this course to improve.



Team

Lecturers:

Jacco Bikker

bikker.j@gmail.com / j.bikker@uu.nl

Office: BBL 424

Debabrata Panja

d.panja@uu.nl

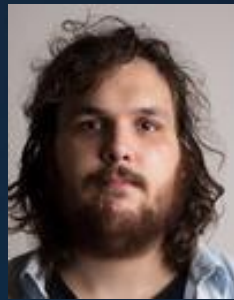
Office: BBL 511



Team

Student Assistants:

1. Kevin van Mastrigt
2. Sander Vanheste
3. Zino Onomiwo
4. Hugo Hogenbirk
5. Niek Mulleners



Practical Details

Assignment Overview:

- P1: “Tutorial”;
- P2: Ray Tracing;
- P3: Rasterization.

Final practicum grade is $0.2 * P1 + 0.4 * P2 + 0.4 * P3$.

Exam overview:

- T1: Mid-term exam;
- T2: Final exam.

Final exam grade is $0.3 * T1 + 0.7 * T2$.

Final grade: $(T + P) / 2$

Passing criteria:

Final Grade ≥ 6 (after rounding); both T and P ≥ 5.0 (before rounding).



Practical Details

How to hand in assignments:

- <http://www.cs.uu.nl/docs/submit>

First assignment (“Tutorial”) is online now:
See website.



Practical Details

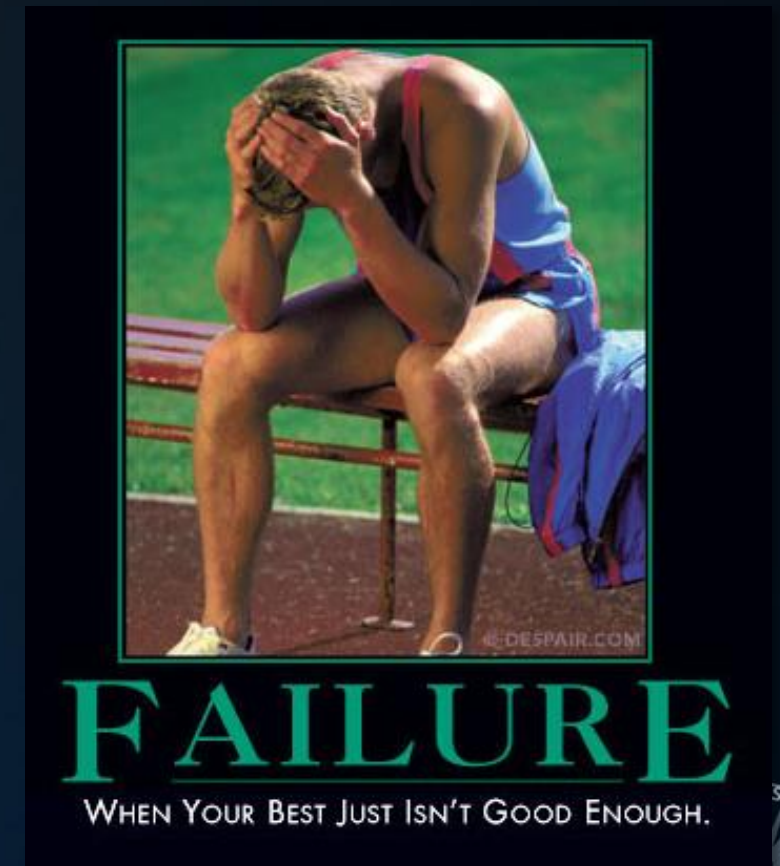
Retake: only if you failed the course, and scored at least a 4.0 (before rounding).

Retake / Theory:

- Retake covers all theory and replaces $\min(T1, T2)$.

Retake / Practical:

- Retake replaces $\min(P1, P2, P3)$.
Topic will be assigned individually.



Assignments

PART 1: Mathematics

Tutorial 1 will be available on Thursday, April 28th.

PART 2: Programming assignment

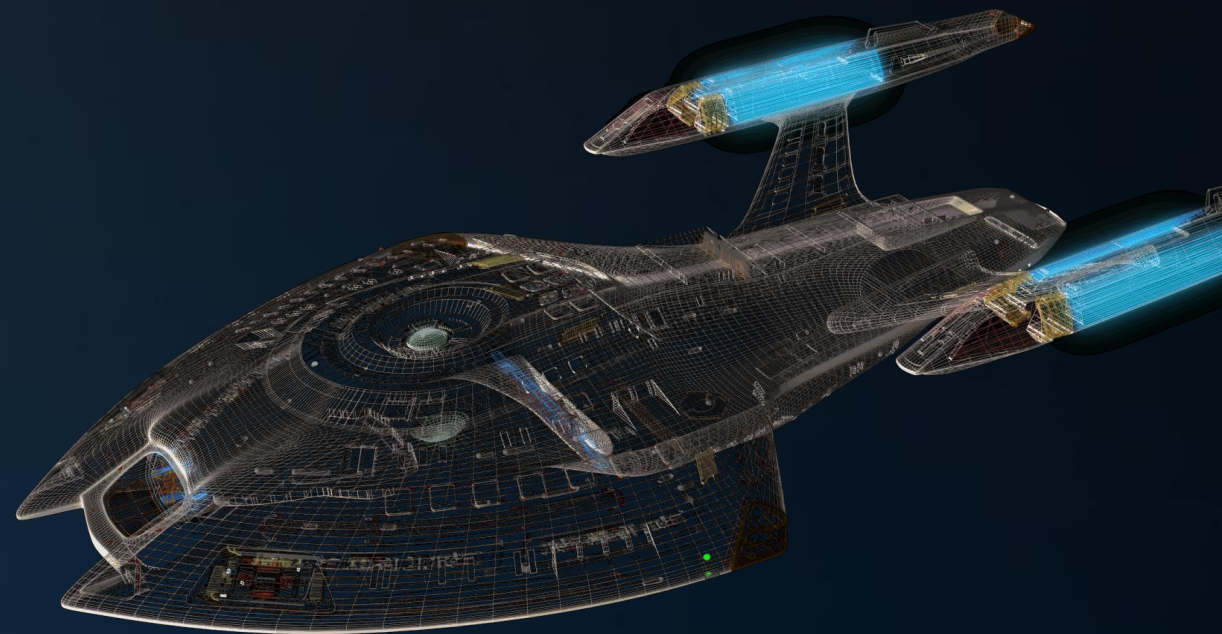
P1 (OpenTK Tutorial) is now available from the website.

Assistance is available on Tuesday, May 3rd in rooms
BBG-079, -083, -109 and -112.

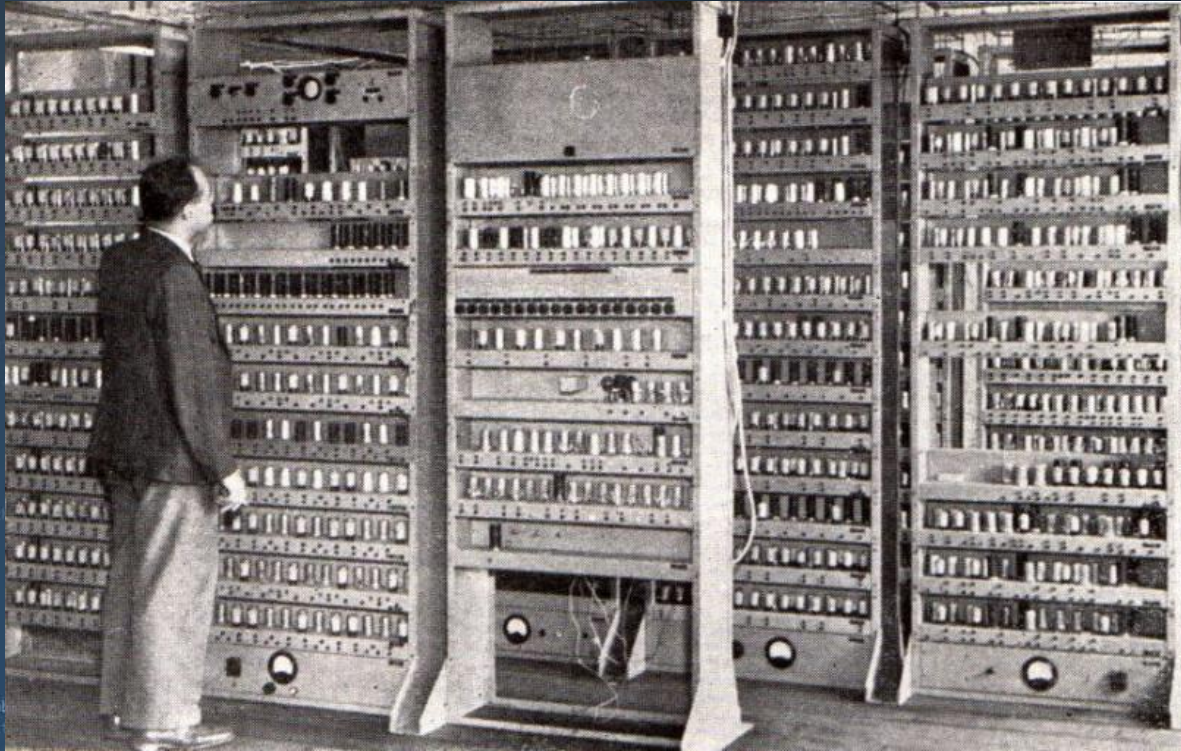


Today's Agenda:

- Graphics
- Course Introduction
- Field Study
- Rasters
- Colors



Field Study



A. S. Douglas. Noughts and Crosses. EDSAC, 1952.

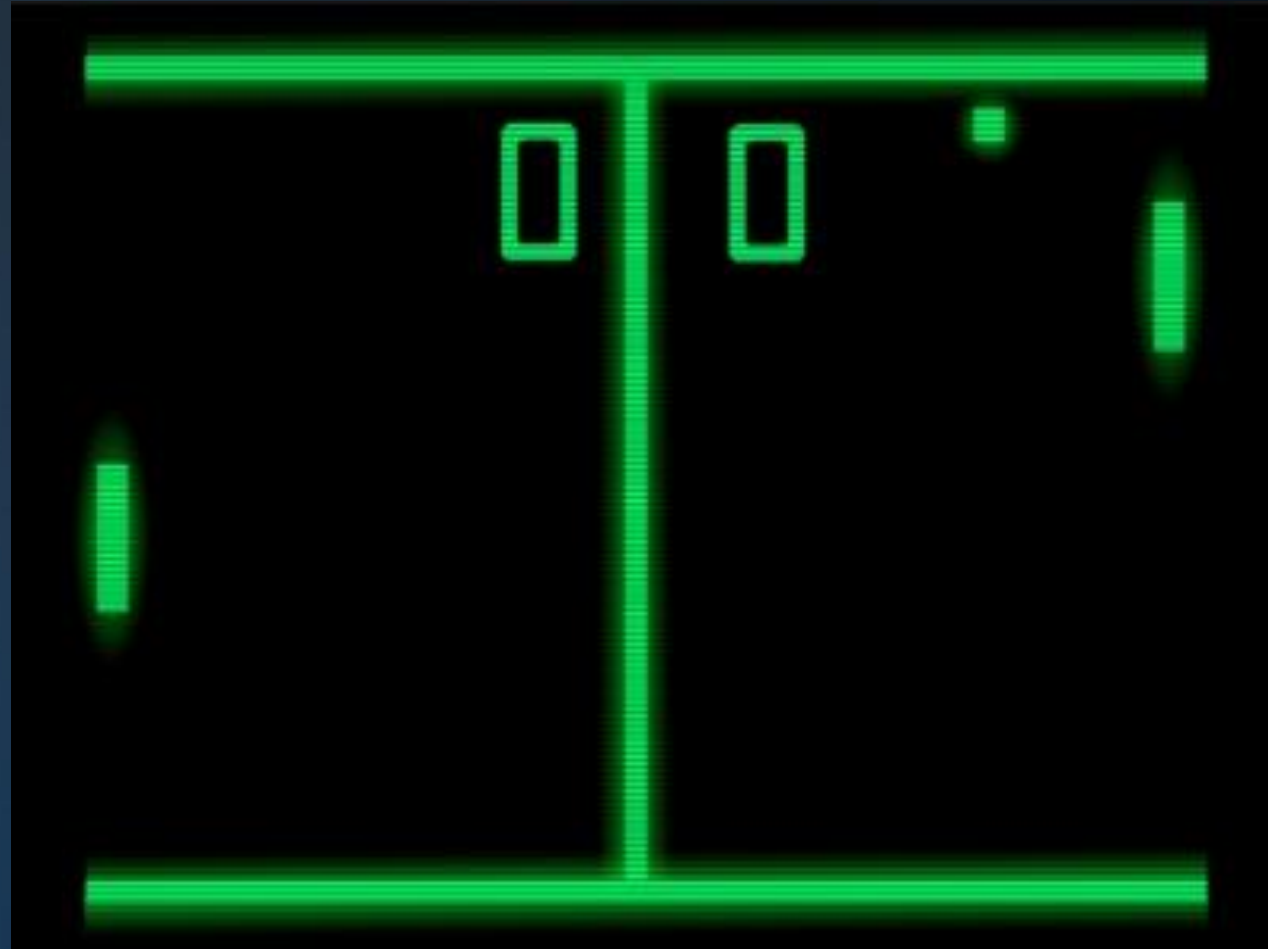


Field Study

```

    if (depth < MAXDEPTH) {
        // Inside the object
        Vec inside = L;
        Vec nt = nt / nc, nde = nde / nc;
        Vec pos2t = 1.0f - nnt * nnt;
        Vec D, N;
        Vec R = (D * nnt - N * nde);
        Vec E * diffuse;
        bool = true;
        Vec refl + refr)) && (depth < MAXDEPTH) {
            Vec D, N;
            Vec refl * E * diffuse;
            bool = true;
        }
    }
    Vec survive = SurvivalProbability( diffuse );
    Vec estimation - doing it properly, closely following walls (survive)
    if (survive) {
        Vec radiance = SampleLight( &rand, I, &t, &light );
        Vec e.x + radiance.y + radiance.z > 0) && (e.x + radiance.y + radiance.z > 0) && (e.x + radiance.y + radiance.z > 0) {
            Vec v = true;
            Vec brdfPdf = EvaluateDiffuse( L, N );
            Vec factor = diffuse * INVPI;
            Vec weight = Mis2( directPdf, brdfPdf );
            Vec cosThetaOut = dot( N, L );
            Vec E * ((weight * cosThetaOut) / directPdf) * (radiance);
        }
    }
    Vec random walk - done properly, closely following walls (survive)
    Vec survive)
    Vec;
    Vec brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    Vec survive;
    Vec pdf;
    Vec n = E * brdf * (dot( N, R ) / pdf);
    Vec n = true;

```



Field Study



1981



1982



1982



Field Study



1985



1987



1990

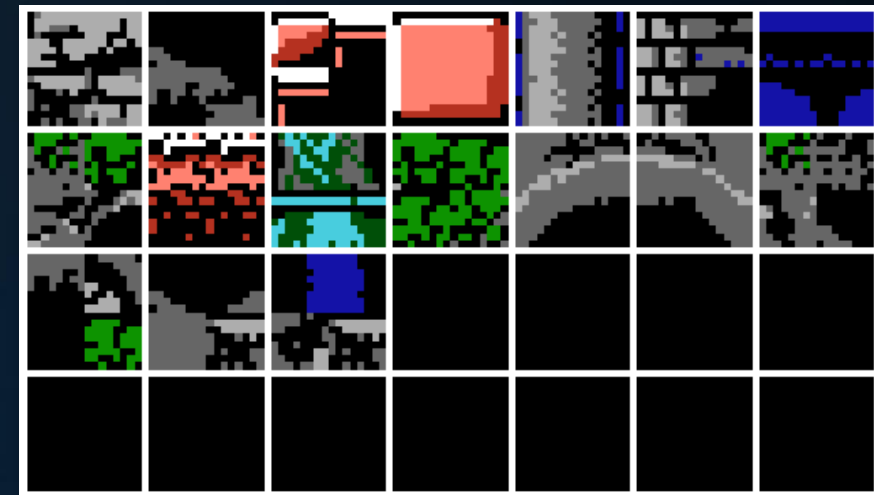
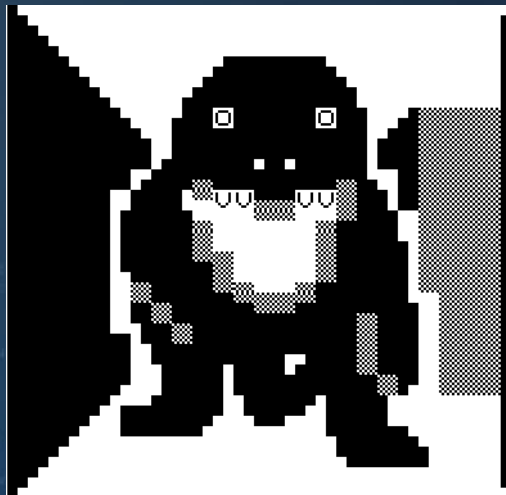


Field Study

Early graphics:

2D, with limitations

- Tiles
- Few colors
- Sprites



Field Study



```
at brdfPdf = EvaluateDiffuse( L, N ) = Pdf;
at3 factor = diffuse * INVPI;
```

```
at weight = Mix2( directPdf, brdfPdf );
```

```
at cosThetaOut = dot( N, L );
```

```
E * ((weight * cosThetaOut) / directPdf) * (radiance
```

```
andom walk - done properly, closely following wall (or
```

```
ive)
```

```
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, BR, &pdf );
```

```
urvive;
```

```
pdf;
```

```
n = E * brdf * (dot( N, R ) / pdf);
```

```
sion = true;
```







THE FLIP Channel

A500

OPERATIONS
MANUAL

AMIGA

Commodore 1084X
VIDEO MONITOR



TODO:

- BUY SHIP
- FIND BIG WOOD
- DEFEAT LECHUCK
- MARRY ELAINE!

Paul O'Brien, 2020

VIDEO: 1084X



50

AMMO

100%

HEALTH

2

3

4

5

6

7

ARMS



2%

ARMOR

BULL
SHEL
ROCK
CELL

50
0
0
0

200
50
50
300





History of Graphics

```
...ics
& (depth < MAXDEPTH)
{
    nt = inside / 1.5;
    nt = nt / nc; add = 1.0f - nt;
    pos2t = 1.0f - nt;
    D, N );
}

at a = nt - nc; b = nt - nc;
at Tr = 1 - (RB + (1 - RB) *
Tr) R = (D * nt - N * (add

E * diffuse;
= true;

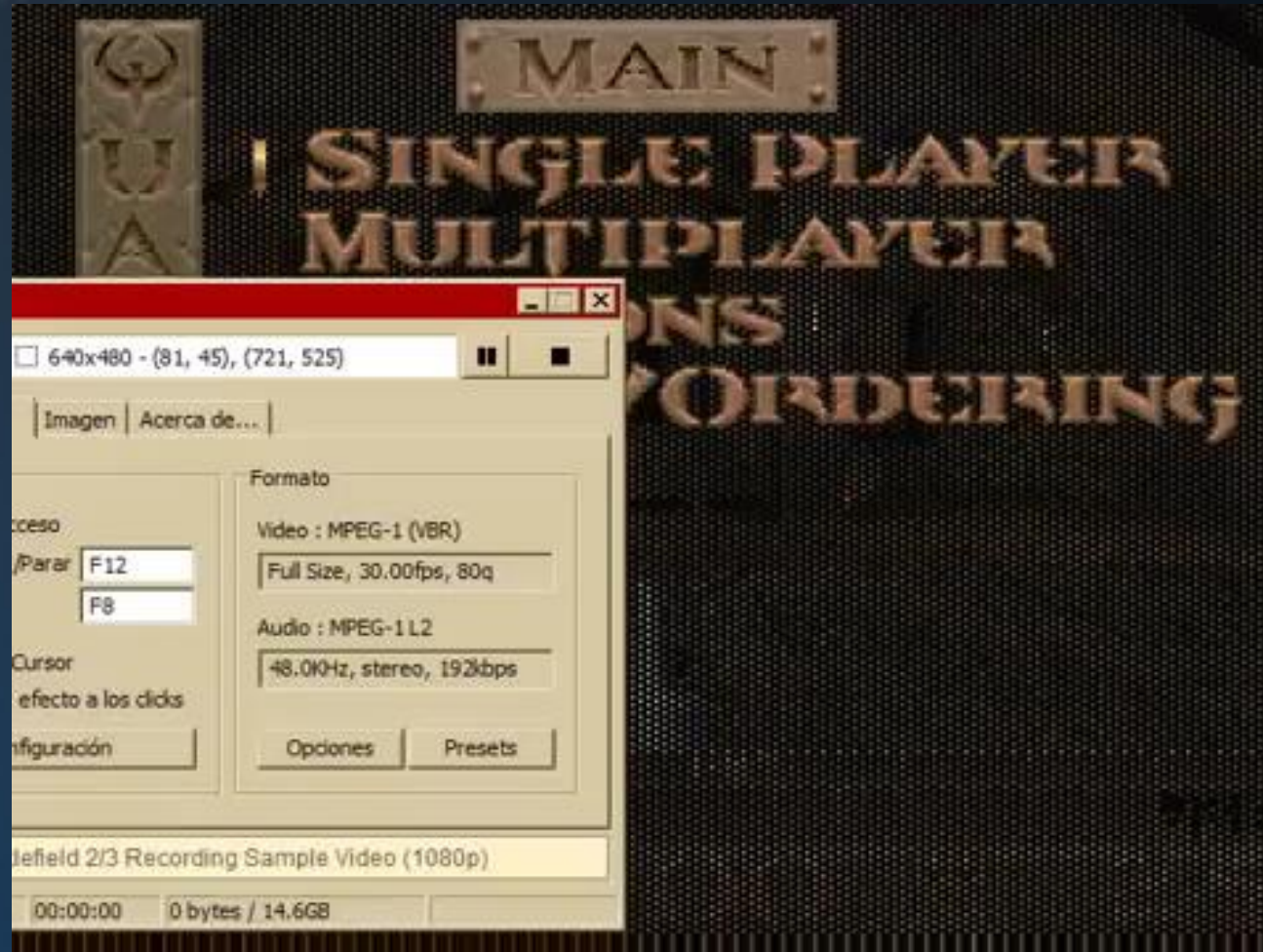
efl + refr)) && (depth < MAXDEPTH)
{
    D, N );
    refl * E * diffuse;
    = true;
}

MAXDEPTH)

survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following walk
if;
radiance = SampleLight( &rand, I, &t, &align,
e.x + radiance.y + radiance.z ) > 0) && (depth <
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
at3 factor = diffuse * INVPI;
at weight = Mix2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiant

random walk - done properly, closely following walk
ive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

















Field Study

Game production:

Code
Art

Crysis:

> 1M lines of code; 85k shaders

Unreal 3 engine:

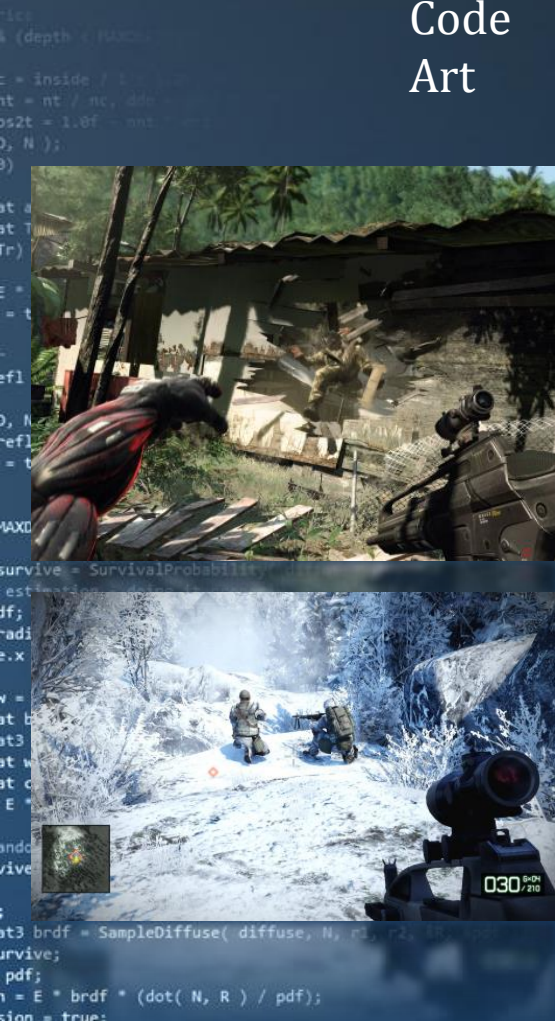
2M lines of code

Frostbite:

“10x Unreal 3”

Minecraft:

< 200k lines of code.



Field Study

History of graphics in games, digest

Initially fast progression:

- from 2D to 3D,
- from monochrome to true-color,
- from wireframe to shaded,
- from sparse to highly detailed.

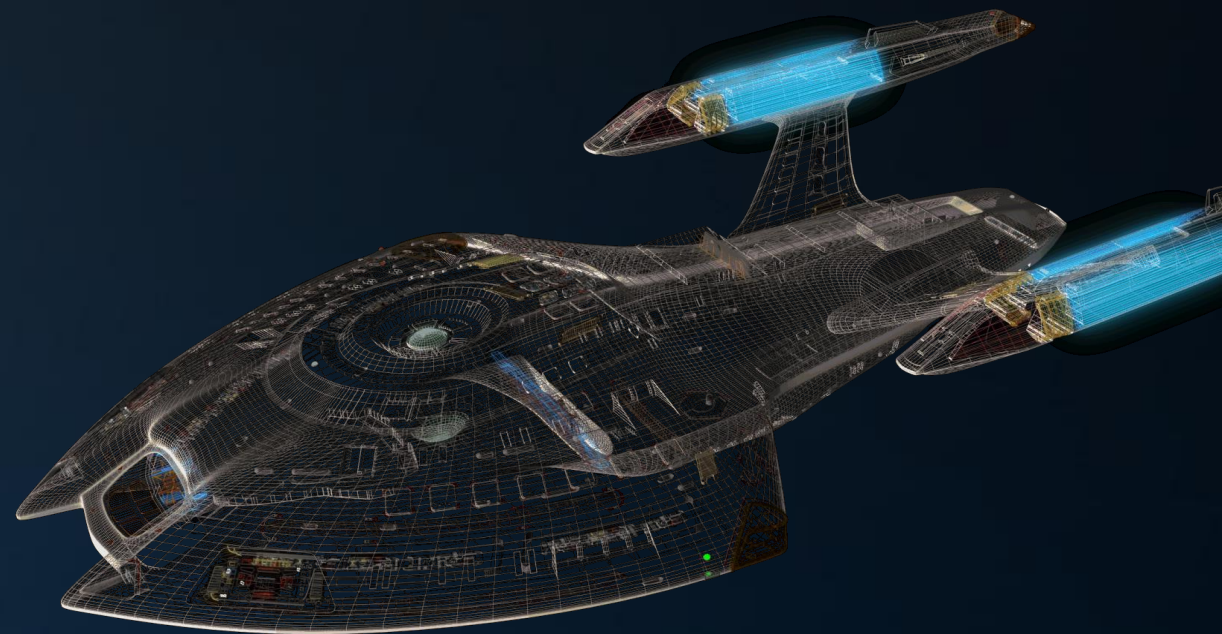
But also:

- from reasonably efficient to produce to extremely labor-intensive.

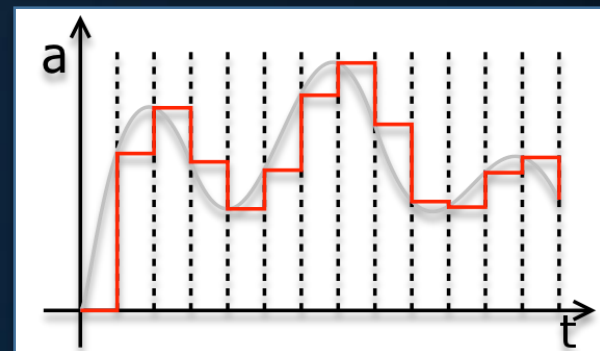
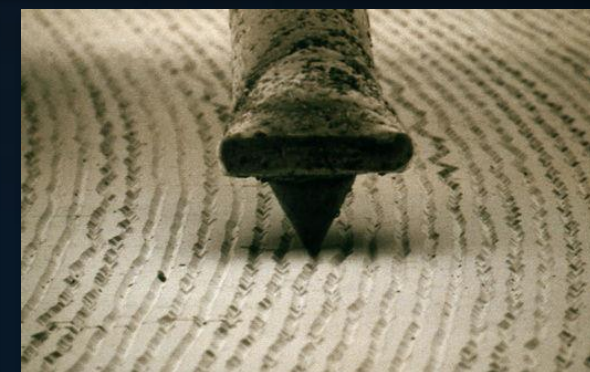
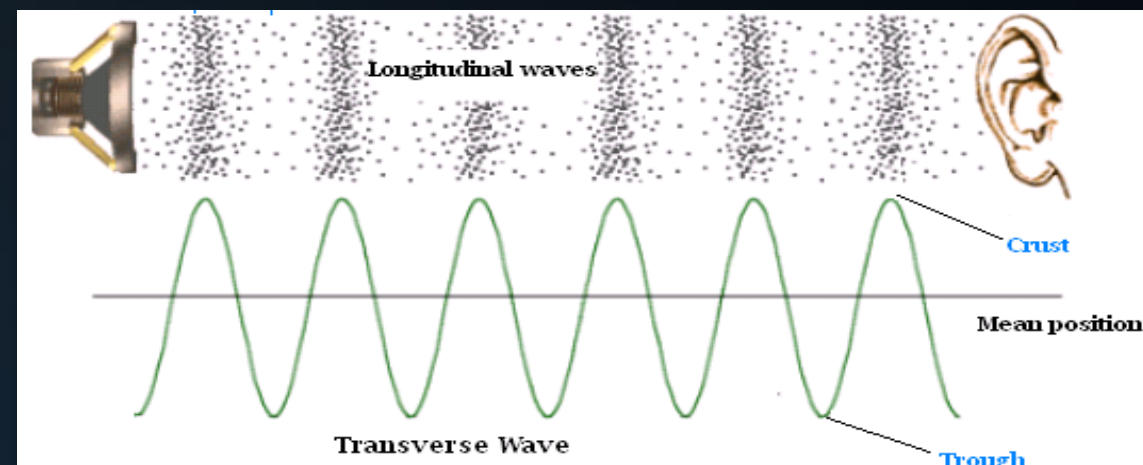
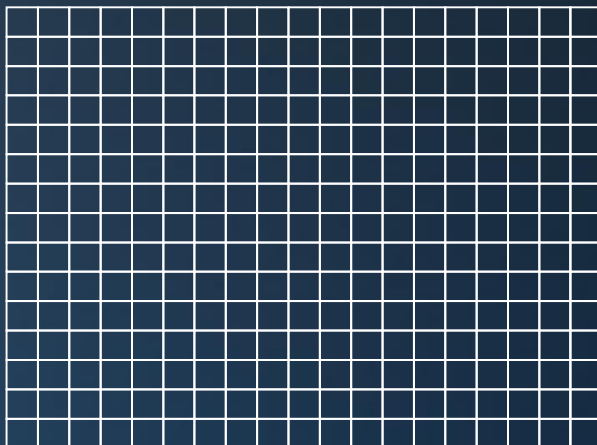


Today's Agenda:

- Graphics
- Course Introduction
- Field Study
- Rasters
- Colors



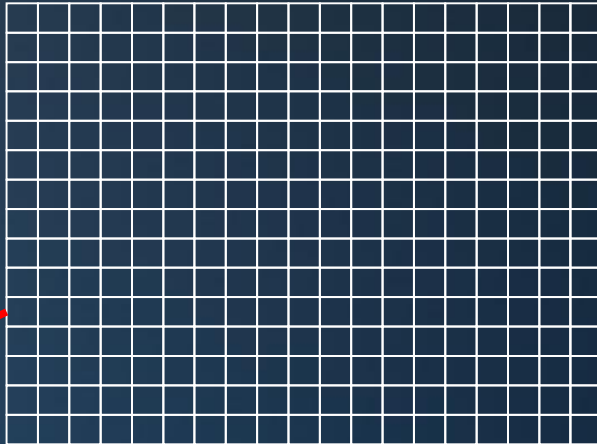
Discretization



Raster Displays

Discretization

```
void RayTracer::Render() {
    // Render the scene
    for (int y = 0; y < HEIGHT; y++) {
        for (int x = 0; x < WIDTH; x++) {
            // Ray casting for each pixel
            Ray r(x, y, 0);
            Color c = r.Trace();
            // Store the color in the image buffer
            image[y][x] = c;
        }
    }
}
```



Rasterization:

“Converting a vector image into a raster image for output on a video display or printer or storage in a bitmap file format.”

(Wikipedia)

```
void RayTracer::Render() {
    // Render the scene
    for (int y = 0; y < HEIGHT; y++) {
        for (int x = 0; x < WIDTH; x++) {
            // Ray casting for each pixel
            Ray r(x, y, 0);
            Color c = r.Trace();
            // Store the color in the image buffer
            image[y][x] = c;
        }
    }
}
```



Raster Displays

Rasterization

```

ices
k (depth < MAXDEPTH)
{
    t = inside / 1.5;
    nt = nt / nc; add = 1.0f - nt;
    pos2t = 1.0f - nnt;
    D, N );
    )
}

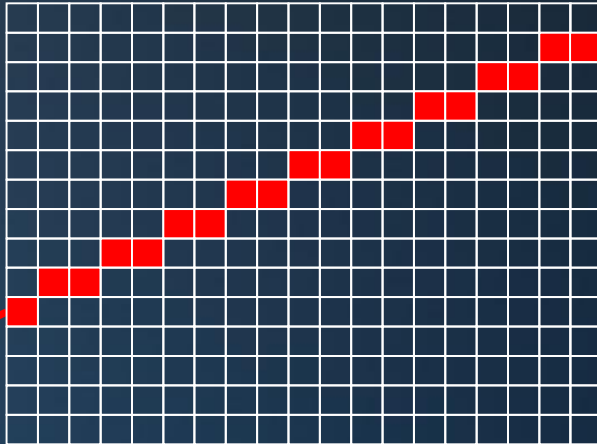
at a = nt - nc, b = nt - nc;
at Tr = 1 - (RB + (1 - RB) * t);
Tr) R = (D * nnt - N * (add

E * diffuse;
= true;

-
efl + refr)) && (depth < MAXDEPTH)
{
    D, N );
    refl * E * diffuse;
    = true;
}

MAXDEPTH)

```



Improving rasterization:

1. Increase resolution;

```

survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &t, &light );
e.x + radiance.y + radiance.z) > 0) && (depth <

```

```

v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance

```

```

random walk - done properly, closely following
ive)

```

```

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



Raster Displays

Rasterization

```
...ics  
if (depth < MAX_DEPTH)
```

```
...t = inside / 1.5f;  
nt = nt / nc; add = ...  
os2t = 1.0f - nnt * ...  
D, N );  
);
```

```
...at a = nt - nc, b = nt - nc;  
...at Tr = 1 - (R0 + (1 - R0) * ...  
...Tr) R = (D * nnt - N * ...
```

```
...E * diffuse;  
...= true;
```

```
...efl + refr)) && (depth < MAX_DEPTH)
```

```
...D, N );  
...refl * E * diffuse;  
...= true;
```

```
...MAX_DEPTH)
```

```
...survive = SurvivalProbability( diffuse );  
...estimation - doing it properly, closely following walk
```

```
...if;
```

```
...radiance = SampleLight( &rand, I, &t, &light);
```

```
...e.x + radiance.y + radiance.z) > 0) && (rand.N < ...
```

```
...v = true;
```

```
...at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
```

```
...at3 factor = diffuse * INVPI;
```

```
...at weight = Mis2( directPdf, brdfPdf );
```

```
...at cosThetaOut = dot( N, L );
```

```
...E * ((weight * cosThetaOut) / directPdf) * (radiance
```

```
...random walk - done properly, closely following walk
```

```
...ive)
```

```
...;
```

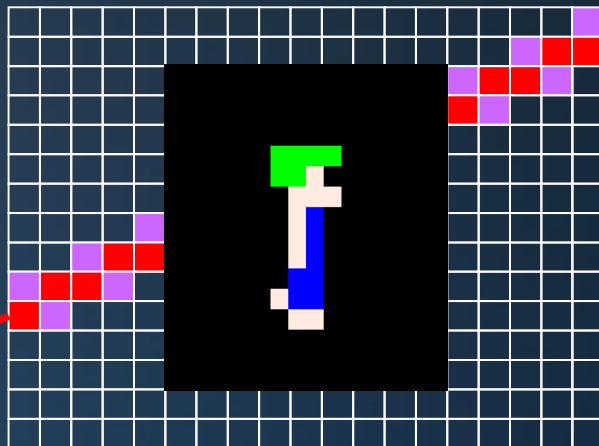
```
...at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
```

```
...survive;
```

```
...pdf;
```

```
...n = E * brdf * (dot( N, R ) / pdf);
```

```
...ision = true;
```



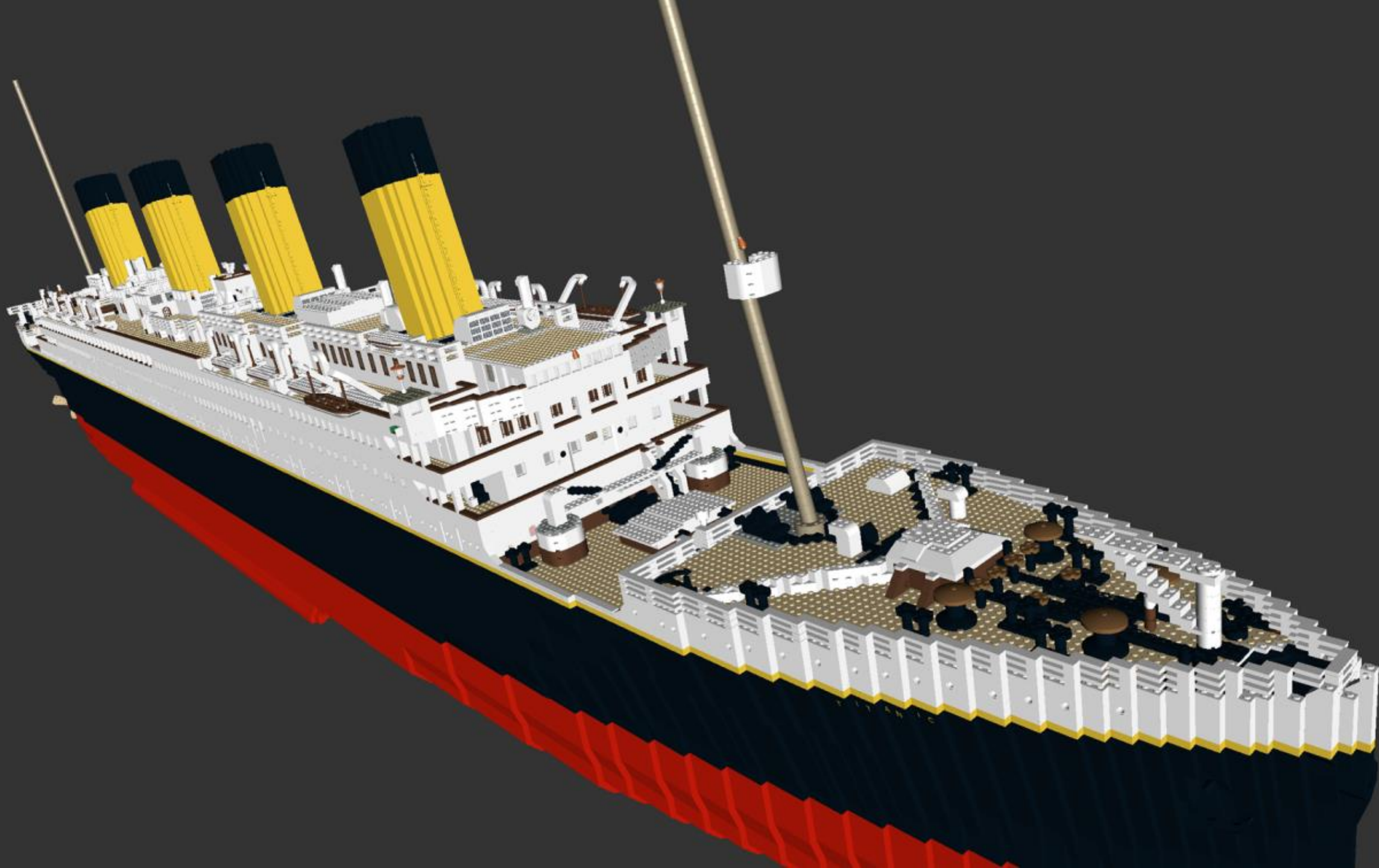
Improving rasterization:

1. Increase resolution;
2. Anti-aliasing;
3. Animation.

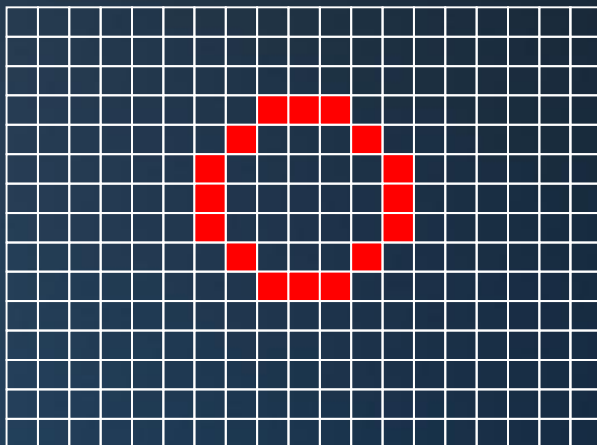








Discretization



$$\pi = 4$$

~~$$a^2 + b^2 = c^2$$~~

$$a^1 + b^1 = c^1$$



Raster Displays

CRT – Cathode Ray Tube

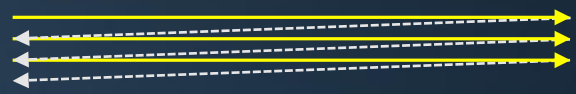
```
...ics
& (depth < MAXDEPTH)
{
    // Inside the sphere
    t = inside / 1.5;
    nt = nt / nc; addo = addo / nc;
    cos2t = 1.0f - nnt * nnt;
    if (cos2t < 0) cos2t = 0;
    D, N );
}

// Ray-sphere intersection
at a = nt - nc, b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * t);
Tr) R = (D * nnt - N * (addo
// Ray-sphere intersection
E * diffuse;
= true;
}

// Ray-sphere intersection
efl + refr)) && (depth < MAXDEPTH)
{
    D, N );
    refl * E * diffuse;
    = true;
}

// Ray-sphere intersection
MAXDEPTH)
{
    survive = SurvivalProbability( diffuse );
    // Russian roulette estimation - doing it properly, closely following well known
    if (survive)
    {
        radiance = SampleLight( &rand, I, &t, &light, &N );
        e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
    {
        v = true;
        at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
        at3 factor = diffuse * INVPI;
        at weight = Mis2( directPdf, brdfPdf );
        at cosThetaOut = dot( N, L );
        E * ((weight * cosThetaOut) / directPdf) * (radiance
    }

    // Random walk - done properly, closely following well known
    survive)
    {
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        ion = true;
    }
}
```



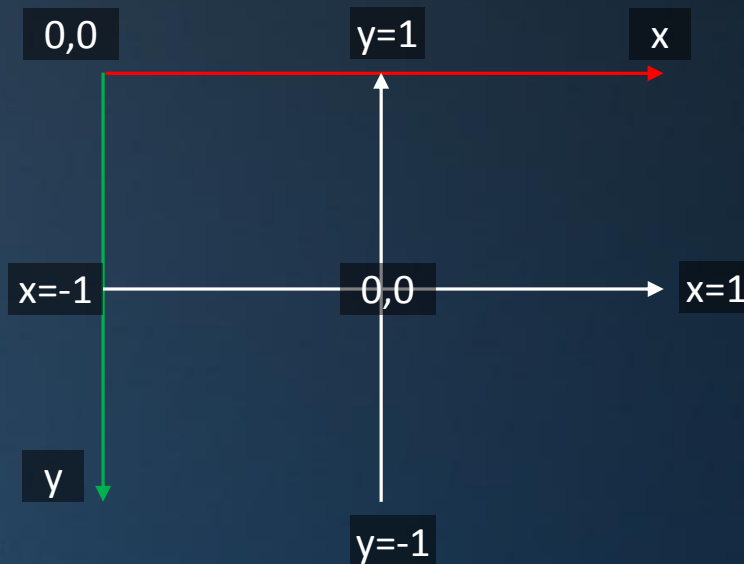
Physical implementation – origins

Electron beam zig-zagging over a fluorescent screen.



Raster Displays

CRT – Cathode Ray Tube



Physical implementation – consequences

- Origin in the top-left corner of the screen
- Axis system directly related to pixel count

Not the coordinate system we expected...



Raster Displays

Frame rate



PAL: 25fps

NTSC: 30fps (actually: 29.97)

Typical laptop screen: 60Hz

High-end monitors: 120-240Hz

Cartoons: 12-15fps

Human eye:

‘Frame-less’

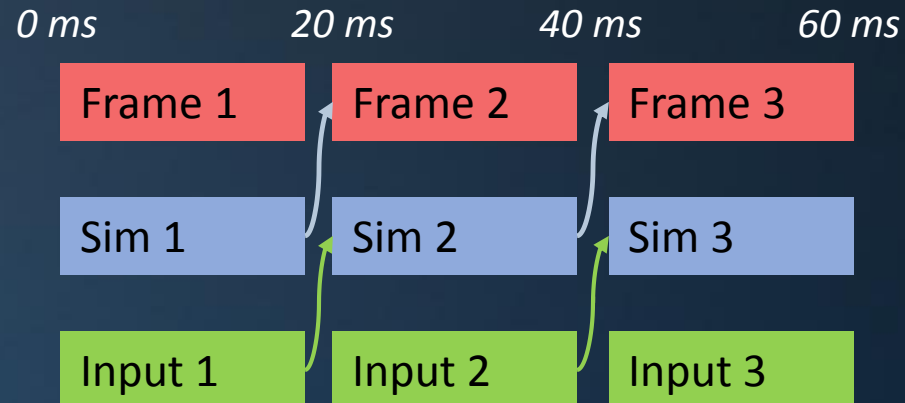
Not a raster.

How many fps / megapixels is ‘enough’?



Raster Displays

Frame rate



Even 100 frames per second may result in a noticeable delay of 30ms.

A very high frame rate minimizes the response time of the simulation.



Raster Displays

Generating images

Rendering:

on a raster

“The process of generating an image from a 2D or 3D model by means of a computer program.”
(Wikipedia)

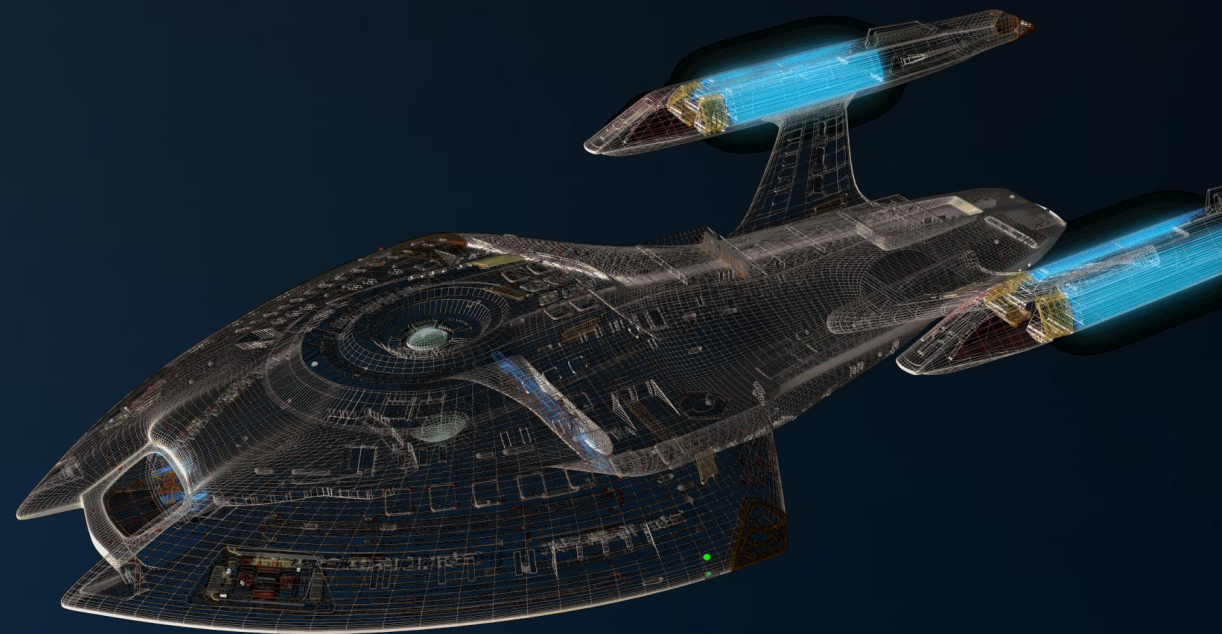
Two main methods:

1. Ray tracing: for each pixel: what color do we assign to it?
2. Rasterization: for each triangle, which pixels does it affect?



Today's Agenda:

- Graphics
- Course Introduction
- Field Study
- Rasters
- Colors



Colors

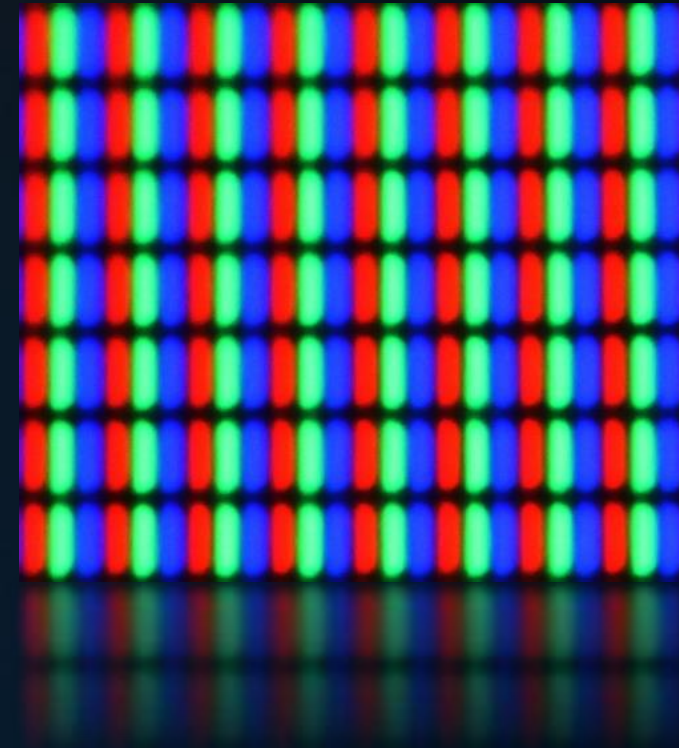
Color representation

Computer screens emit light in three colors: red, green and blue.

By additively mixing these, we can produce most colors: from black (red, green and blue turned off) to white (red, green and blue at full brightness).

In computer graphics, colors are stored in discrete form. This has implications for:

- Color resolution (i.e., number of unique values per component);
- Maximum brightness (i.e., range of component values).



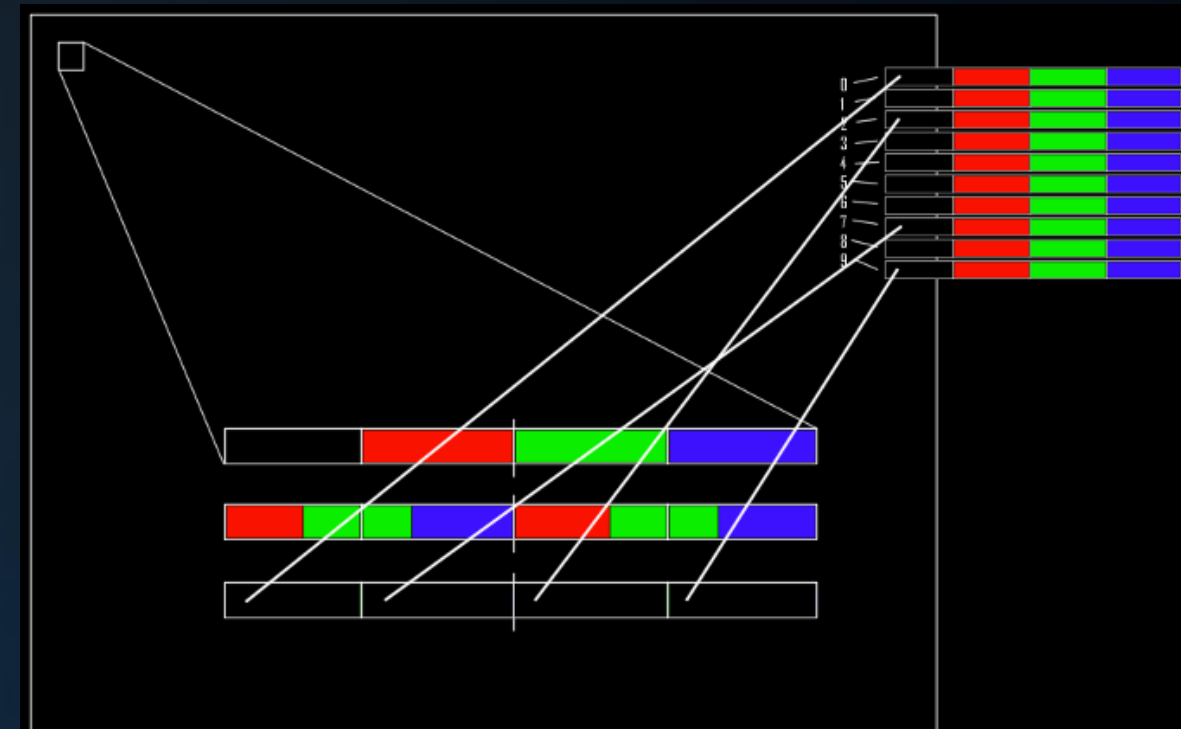
Colors

Color representation

The most common color representation is 32-bit ARGB, which stores red, green and blue as 8 bit values (0..255).

Alternatively, we can use 16 bit for one pixel (RGB 565),

or a color palette. In that case, one byte is used per pixel, but only 256 unique colors can be used for the image.



Colors

Color representation

```
...
    & (depth < MAXDEPTH)
{
    // Inside the sphere
    nt = inside / 1.5;
    nc = nt * nc;
    ndd = nt * (1 + nc);
    n2t = nt * (1 - nc);
    r2t = 1.0f - n2t;
    r = (D * N);
    R = (D * nnt - N * (ndd * r2t));
    // Diffuse reflection
    E * diffuse;
    // Refractive index
    refl + refr)) && (depth < MAXDEPTH)
{
    // Inside the sphere
    D, N );
    refl * E * diffuse;
    // Refractive index
    MAXDEPTH)
{
    survive = SurvivalProbability( diffuse );
    // Estimation - doing it properly, closely following walk
    if;
    radiance = SampleLight( &rand, I, &lt;, &light; );
    // Radiance
    e.x + radiance.y + radiance.z > 0) && (depth < MAXDEPTH)
{
    // Inside the sphere
    w = true;
    // Evaluate diffuse
    at brdfPdf = EvaluateDiffuse( L, N ); * Psurvive;
    // Factor
    at3 factor = diffuse * INVPI;
    // Weight
    at weight = Mis2( directPdf, brdfPdf );
    // CosThetaOut
    at cosThetaOut = dot( N, L );
    // Radiance
    E * ((weight * cosThetaOut) / directPdf) * (radiance);
    // Random walk - done properly, closely following walk
    survive)
{
    // Evaluate diffuse
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    // Survive
    survive;
    // Pdf
    pdf;
    // Radiance
    n = E * brdf * (dot( N, R ) / pdf);
    // Estimation
    sion = true;
}
```



True color (24bit)

Colors

Color representation

```
...
    & (depth < MAXDEPTH)
{
    // Inside the sphere
    nt = inside / 1.5;
    nc = nt * nc;
    ndd = nt * (1 + nc);
    n2t = nt * 1.0f - ndd;
    nt = 0;
    ndd = 0;
    N = (D * N);
    N = N * ndd;
    R = (D * N);
    R = R * ndd;
    E * diffuse;
    = true;
    refl + refr)) && (depth < MAXDEPTH)
{
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &align,
e.x + radiance.y + radiance.z) > 0) && (depth <
w = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
at3 factor = diffuse * INVPI;
at weight = Mix2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
random walk - done properly, closely following
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
ision = true;
```



Hicolor (16bit)

Color representation

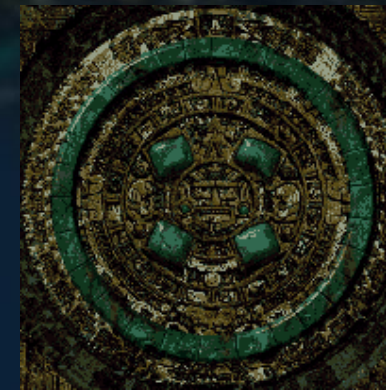
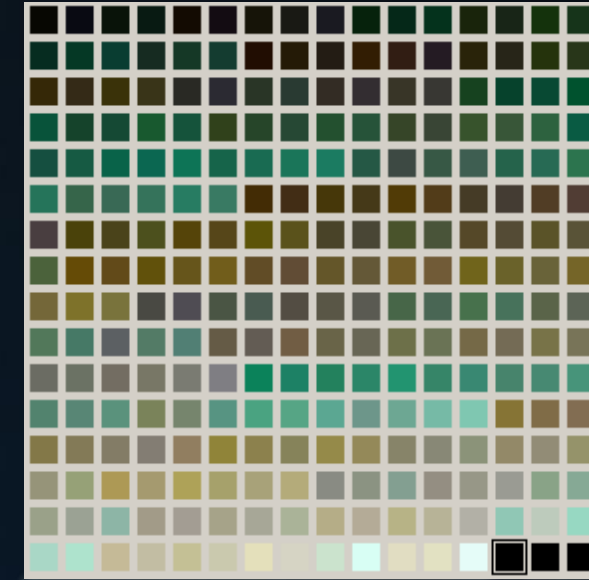


Colors

Color representation

Textures can typically safely be stored as palletized images.

Using a smaller palette will result in smaller compressed files.



Colors

Color representation

Using a fixed range (0:0:0 ... 255:255:255) places a cap on the maximum brightness that can be represented:

- A white sheet of paper: (255,255,255)
- A bright sky: (255,255,255)

The difference becomes apparent when we look at the sky and the sheet of paper through sunglasses.

(or, when the sky is reflected in murky water)



Colors

Color representation

For realistic rendering, it is important to use an internal color representation with a much greater range than 0..255 per color component.

HDR: High Dynamic Range;

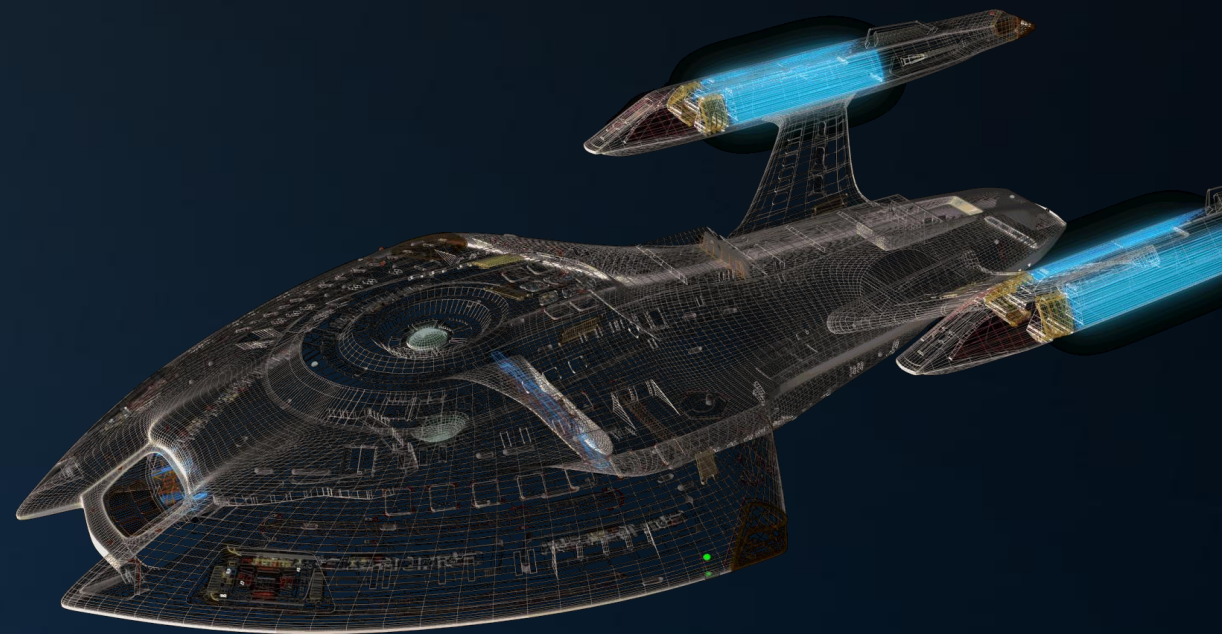
We store one float value per color component.

Including alpha, this requires 128bit per pixel.



Today's Agenda:

- Graphics
- Course Introduction
- Field Study
- Rasters
- Colors



Next Time:

- Ray Tracing
 - Primitives
 - Vectors
 - Coordinate systems
 - Projections



INFOGR – Computer Graphics

Jacco Bikker & Debabrata Panja - April-July 2017

END OF lecture 1: “Introduction”



Next lecture: “Ray Tracing”

