tic: ⊾ (depth ⊂ PSs

: = inside / l nt = nt / nc, dd os2t = 1.0f - on 0, N ); 0)

at a = nt - nc, b - nt at Tr = 1 - (80 + (1 Tr) R = (0 \* nnt - N

E <sup>=</sup> diffuse = true;

-:fl + refr)) && (depth is HADDIE

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; adiance = SampleLight( %rand, I, M) e.x + radiance.y + radiance.z) > 0)

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Puncture st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* 100

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, R, R, r pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

# INFOGR – Computer Graphics

Jacco Bikker & Debabrata Panja - April-July 2017

## Lecture 7: "Accelerate"

# Welcome!



## Recap



- - Anside it = nt / nc, dde os2t = 1.0f - nnt O, N ); B)

st a = nt - nc, b - nt - st Tr = 1 - (R0 + (1 - 1 fr) R = (D \* nnt - N

= diffuse; = true;

: :fl + refr)) 88 (depth k MAXDIII

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property if; radiance = SampleLight( &rand, I e.x + radiance.y + radiance.r) = 0

w = true; at brdfPdf = EvaluateDiffuse( L, N ) \* F st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf );

at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* (PD

andom walk - done properly, closely following : /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, brdf prvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true;





PC TO XED Mere TA

0.

## Recap

), N ); efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( dif if; adiance = SampleLight( &rand, I. e.x + radiance.y + radiance.z) > 0)

v = true;

at brdfPdf = EvaluateDiffuse( L, N ) st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ) at cosThetaOut = dot( N, L );

E ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely follo vive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, N rvive; pdf; i = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## Color Trace( vec3 0, vec3 D ) I, N, mat = IntersectScene( 0, D ); if (!I) return BLACK; return DirectIllumination( I, N ) \* mat.diffuseColor;



 $Ax + By + Cz + d = \widehat{N} \cdot p + d = 0$  and  $p(t) = 0 + t\widehat{D}$  $\widehat{N} \cdot (O + t\widehat{D}) + d = 0 \Rightarrow \widehat{N} \cdot O + \widehat{N} \cdot (t\widehat{D}) + d = 0$  $\widehat{N} \cdot (t\widehat{D}) = -(\widehat{N} \cdot O + d) \rightarrow t = -(\widehat{N} \cdot O + d)/(\widehat{N} \cdot \widehat{D})$ 



 $p_0$ 

 $p_2$ 

 $p_2$ 

 $p_1 - p_0$ 

 $p_1$ 

E



Camera looks straight ahead along z:



 $C = E + d\hat{V}$ 



Arbitrary  $\hat{V}$ :

 $p_0$ 

 $p_0$ 

 $p_2$ 

 $p_1 - p_0$ 

 $\begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}$ 

С

 $p_0 = C - \hat{r} + \hat{u}$ 

 $p_1 = \mathcal{C} + \hat{r} + \hat{u}$ 

 $p_2 = C - \hat{r} - \hat{u}$ 

0

 $p_1$ 

E

let  $\widehat{up} = \begin{pmatrix} 0\\1\\0 \end{pmatrix}$ . Then:





tic: ≰ (depth < 10.55

1 = 105108 / 1 1t = nt / nc, dde 052t = 1.0f - nnt 2, N ); 3)

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

= diffuse; = true;

: :fl + refr)) && (depth < H

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)



tica ≰ (depth < PAx

= inside / 1 it = nt / nc, dda os2t = 1.0f - nnt 0, N(); 3)

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

= diffuse; = true;

-: efl + refr)) && (depth k HANDIIII

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; adiance = SampleLight( &rand, I, I, e.x + radiance.y + radiance.z) > 0) ##

v = true; t brdfPdf = EvaluateDiffuse( L, N ) Promote st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, brd pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## Today's Agenda:

- Efficiency
- Boxes, AABBs & Groupings
- To Rasterization
- 3D Engine Overview
- Textures





# Efficiency

tice ⊾ (depth is 1925

= inside / 1 it = nt / nc, ddo ss2t = 1.0f - nnt -), N ); 3)

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

= diffuse; = true;

-:fl + refr)) && (depth is HANDIII

D, N ); ~efl \* E \* diffuse; = true;

AXDEPTH)

e.x + radiance.y + radiance.z) > 0) 88 (constant) v = true;

at brdfPdf = EvaluateDiffuse( L, N ) Power st3 factor = diffuse = INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following : /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, s); pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## **Measuring Performance**

## Stopwatch class:

Using System.Diagnostics.Stopwatch;

## Useful property:

long ElapsedMilliseconds { get; }

### Methods:

- Reset
- Start
- Stop

### Note:

Accuracy may vary. Measure lots of work, not a single line of code. Aim for tens of milliseconds, not nanoseconds.

### Note:

Multithreading affects measurements. Profile single-threaded code; tune your multi-threading independently.

### Note:

Use a profiler for more accuracy and detail. Try e.g. SlimTune, or Prof-It: <u>http://prof-it.sourceforge.net</u>



## 7

# Efficiency

tic: k (depth < 100

: = inside / l it = nt / nc, ddo os2t = 1.0f - nnt -O, N ); 3)

st a = nt - nc, b - nt - --st Tr = 1 - (R0 + (1 - ---fr) R = (0 \* nnt - N

= diffuse; = true;

-•fl + refr)) 88 (death ( 1000)

D, N ); ~efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property ff; radiance = SampleLight( %rand, I e.x + radiance.y + radiance.z) > 0) %

#### v = true;

st brdfPdf = EvaluateDiffuse( L, N.) \* Pauro st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

indom walk - done properly, closely following
vive)

st3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, state urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true;

## **Optimization Primer**

Some things to keep in mind:

- Float or double
- Don't do work you don't need to do
  - Early out
  - Reduce precision
  - Lights with finite radius
  - Things that can't occlude light





# Efficiency

tice k (depth < RAS

= inside / l it = nt / nc, dde ss2t = 1.0f - nnt -), N ); 3)

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

= diffuse; = true;

≕ efl + refr)) 88 (depth k HANDIII

D, N ); ~efl \* E \* diffuse; = true;

#### AXDEPTH)

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

v = true;

st brdfPdf = EvaluateDiffuse( L, N.) Pauro bit st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* (Pauro E \* (weight \* cosThetaOut) / directPdf) \* (Pauro E \* (weight \* cosThetaOut) / directPdf) \* (Pauro E \* (weight \* cosThetaOut) \* (weight \* cosThet

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, Dpd prvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true;

## **Optimization Primer**

Some things to keep in mind:

- Float or double
- Don't do work you don't need to do
- Precalculate
  - Loop hoisting
  - Vertex shaders

## L.Normalize(); if (dot( L, N ) > 0) { $\rightarrow$ vec3 N = intersection->GetNormal(); for( int i = 0; i < lights; i++ ) {</pre> vec3 L = light[i]->pos intersection->pos; if (dot( L, N ) > 0) { L.Normalize();



# Efficiency

efl + refr)) && (depth is HAN

), N ); refl \* E \* diffuse;

AXDEPTH)

survive = SurvivalProbability( diff adiance = SampleLight( &rand, I, LL e.x + radiance.y + radiance.z) > 0) #

v = true; at brdfPdf = EvaluateDiffuse( L, N ) st3 factor = diffuse \* INVPI: at weight = Mis2( directPdf, brdfPdf ) at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely follow vive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, LR irvive; pdf; i = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## **Optimization Primer**

Some things to keep in mind:

- Float or double
- Don't do work you don't need to do
- Precalculate
- Expensive operations
  - sin, cos
  - sqrt





...

If you need sin/cos, it's often much faster to use a look-up table.

```
float sintab[3600], costab[3600];
for( int i = 0; i < 3600; i++ )</pre>
{
```

```
sintab[i] = Math.Sin( i / 10 );
costab[i] = Math.Cos( i / 10 );
```

```
float s = sintab[(int)(a * 10)];
float c = costab[(int)(a * 10)];
```



# Efficiency

tice ≰ (depth < RAS

= inside / l it = nt / nc, ddo os2t = 1.0f - nnt -D, N ); B)

st a = nt - nc, b - nt st Tr = 1 - (R0 + (1 Tr) R = (D \* nnt - N

= diffuse; = true;

efl + refr)) && (depth & MANDIIII

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; radiance = SampleLight( %rand, I, e.x + radiance.y + radiance.z) > 0) %%

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Pour 1 st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* Context E \* ((weight \* cosThetaOut) / directPdf) \* Context E \* ((weight \* cosThetaOut) / directPdf) \* Context E \* ((weight \* cosThetaOut) / directPdf) \* Context E \* ((weight \* cosThetaOut) / directPdf) \* Context E \* ((weight \* cosThetaOut) / directPdf) \* Context E \* ((weight \* cosThetaOut) / directPdf) \* Context E \* ((weight \* cosThetaOut) / directPdf) \* Context E \* ((weight \* cosThetaOut) / directPdf) \* Context E \* ((weight \* cosThetaOut) / directPdf) \* Context E \* ((weight \* cosThetaOut) \* Context E \* (weight \* cosThetaOut) \* (weight \* cosThetaOut) \* (weight \* cosThetaOut) \* (weight \* cosThetaOut) \* (weig

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, so pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## **Optimization Primer**

Some things to keep in mind:

- Float or double
- Don't do work you don't need to do
- Precalculate
- Expensive operations
- Programming Language
  - C#/C++
  - C++/Asm



# Efficiency

tice 6 (depth is 1925

: = inside / 1 ht = nt / nc, dde os2t = 1.0f - nnt 0, N ); 3)

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

= diffuse; = true;

efl + refr)) && (depth & HADDE

D, N ); refl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; radiance = SampleLight( \$rand, I. 8. 2.x + radiance.y + radiance.z) = 0)

v = true; at brdfPdf = EvaluateDiffuse( L, N) \* Pi st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following : /ive)

, H33 brdf = SampleDiffuse( diffuse, N, F1, F2, RR, F5 prvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## **Perceived Performance**

**Incremental Rendering** 

- 1. Real-time preview:
- Depth map
- Depth map plus materials
- Render without recursive reflections
- Render with very limited recursion

### Still not real-time?

- Render half-res
- Adaptive resolution
  - Optimize the application a bit



# Efficiency

tic: k (depth < 100

: = inside / l ht = nt / nc, dde os2t = 1.0f - nnt / D, N ); B)

at a = nt - nc, b - nt at Tr = 1 - (R0 + (1 - R0 Tr) R = (0 \* nnt - R

= diffuse; = true;

efl + refr)) && (depth k HAADII

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse( L, N ) Pauro st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following : /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, Lor pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## **Perceived Performance**

**Incremental Rendering** 

- 2. Stationary camera:
- Render with normal recursion

Keep the application responsive:

Render lines of pixels until a certain number of milliseconds has passed; continue in the next frame.



# Efficiency

tice ⊾ (depth < 100

: = inside / l ht = nt / nc, ddo os2t = 1.0f - nnt ' o, N ); 3)

at a = nt - nc, b + nt + n at Tr = 1 - (R0 + (1 fr) R = (0 \* nnt - N

= diffuse; = true;

-: :fl + refr)) && (depth is HANDII

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

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

v = true; at brdfPdf = EvaluateDiffuse( L, N) \* Pur st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf); at cosThetaOut = dot( N, L); E \* ((weight \* cosThetaOut) / directPdf)

indom walk - done properly, closely following :
/ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, Lord urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## **Perceived Performance**

**Incremental Rendering** 

- 3. 'Photograph mode':
- Invoked with a key
- Render with extreme recursion
- Use anti-aliasing
- Add screenshot feature

## Keep the application responsive!



tica ≰ (depth < PAx

= inside / 1 it = nt / nc, dda os2t = 1.0f - nnt 0, N(); 3)

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

= diffuse; = true;

-: efl + refr)) && (depth k HANDIIII

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; adiance = SampleLight( &rand, I, I, e.x + radiance.y + radiance.z) > 0) ##

v = true; t brdfPdf = EvaluateDiffuse( L, N ) Promote st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, brd pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## Today's Agenda:

- Efficiency
- Boxes, AABBs & Groupings
- To Rasterization
- 3D Engine Overview
- Textures





## Boxes

tic: k (depth < 100

= inside / L it = nt / nc, dob 552t = 1.0f - nnt 5, N ); 3)

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

= diffuse; = true;

-:fl + refr)) && (depth & HAOLLI

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse( L, W )

st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* (\*)

andom walk - done properly, closely following : /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## **Hierarchical Grouping**

Using AABBs, we can recursively group objects.

- A ray that misses a green box will not check the triangles inside it;
- A ray that misses a blue box will skip the two green boxes inside it;
- A ray that misses the red box doesn't hit anything at all.





## Boxes

tice ≰ (depth < 10.5

: = inside / l ht = nt / nc, dde os2t = 1.0f - nn: 0; N ); 3)

st a = nt - nc, b - nt + + st Tr = 1 - (R0 + (1 Tr) R = (D \* nnt - N

= diffuse = true;

-: :fl + refr)) && (depth is MADDIT

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property if; radiance = SampleLight( &rand I e.x + radiance.y + radiance.r) = 0

v = true;

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

andom walk - done properly, closely following : /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, pp4 pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## Hierarchical Grouping

In a rasterization-based world:

- If a green box is outside the view frustum, we don't have to render the triangles inside it;
- If a blue box is outside the view frustum, we don't have to test the green boxes inside it;
- If the red box is outside the view frustum, we don't see anything.





## Boxes

tic: K (depth < 100

: = inside / l it = nt / nc, ddi os2t = 1.0f - nn ), N ); ))

st  $a = nt - nc_{1}b - nt - st Tr = 1 - (R0 + (1))$ Tr ) R = (D \* nnt - N \* )

= diffuse = true;

: :**fl + refr))** && (depth < MAXD

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse( L, N.) \* Pauro st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* Factored

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, D) pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## Why Do We Care

## We can use an AABB to quickly discard objects.







## Boxes

tice ≰ (depth < 10.5

: = inside / 1 it = nt / nc, dde os2t = 1.0f - nnt 0, N ); 0)

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

= diffuse; = true;

-:fl + refr)) && (depth k HANDIII

D, N ); refl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; radiance = SampleLight( %rand, I, M. e.x + radiance.y + radiance.r) > 0) %

v = true;

st brdfPdf = EvaluateDiffuse( L, N.) Pour st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* (null

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, sourvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## Intersecting a Box

## Basic ray/box intersection:

- 1. Intersect the ray with each of the 6 planes;
- 2. Keep the intersections that are on the same side of the remaining planes;
- 3. Determine the closest intersection point.





## Boxes

tic: ⊾(depth ∈ 100

= inside / 1 it = nt / nc, dda os2t = 1.0f - nnt 0, N(); 3)

st a = nt - nc, b = nt - ncst Tr = 1 - (R0 + (1 - 1))Tr ) R = (D + nnt - N - 1)

= diffuse; = true;

efl + refr)) && (depth x HAOD

D, N ); refl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it property if; radiance = SampleLight( %rand, I &:x + radiance.y + radiance.z) = 0.000

v = true; at brdfPdf :

st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following /ive)

; t3 brdf = SampleDiffuse( diffuse, N, r1, r2, (R, ); pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true;

## Special Case: AABB

AABB: Axis Aligned Bounding Box.

### Slab test:

Intersect the ray against pairs of planes.  $t_{min} = +\infty, t_{max} = -\infty$   $t_{min} = \max(t_{min}, \min(t1, t2))$   $t_{max} = \min(t_{max}, \max(t1, t2))$ intersection if:  $t_{min} < t_{max}$ 

Since the box is axis aligned, calculating t is cheap

 $t = -(0 \cdot \vec{N} + d) / (\vec{D} \cdot \vec{N})$  $= -(O_x \cdot \vec{N}_x + d)/(\vec{D}_x \cdot \vec{N}_x)$  $=(x_{plane}-O_x)/\vec{D}_x$ 



## Boxes

tic: ⊾(depth (192

= inside / 1 + it = nt / nc, dde os2t = 1.0f - nnt -D, N ); B)

st a = nt - nc, b - mt st Tr = 1 - (R0 + (1 Tr) R = (0 \* nnt - N \*

= \* diffuse; = true;

-:fl + refr)) && (depth & MAXDIII

D, N ); refl \* E \* diffuse; = true;

#### AXDEPTH)

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

v = true; t brdfPdf = EvaluateDiffuse( L, N ) Prost3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \*

andom walk - done properly, closely following /ive)

; t33 brdf = SampleDiffuse( diffuse, N, r1, r2, dR, hpd urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## Special Case: AABB

In pseudo-code:

bool intersection( box b, ray r )

float tx1 = (b.min.x - r.0.x) / r.D.x;
float tx2 = (b.max.x - r.0.x) / r.D.x;

float tmin = min( tx1, tx2 );
float tmax = max( tx1, tx2 );

float ty1 = (b.min.y - r.0.y) / r.D.y;
float ty2 = (b.max.y - r.0.y) / r.D.y;

tmin = max( tmin, min(ty1, ty2 ) ); tmax = min( tmax, max(ty1, ty2 ) );

return tmax >= tmin;



tica ≰ (depth < PAx

= inside / 1 it = nt / nc, dda os2t = 1.0f - nnt 0, N(); 3)

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

= diffuse; = true;

efl + refr)) 88 (depth k HANDIII

D, N ); ~efl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; adiance = SampleLight( \$rand, I, I, e.x + radiance.y + radiance.z) > 0) \$\$\$

v = true; at brdfPdf = EvaluateDiffuse( L, N.) Promote st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

indom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, source; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## Today's Agenda:

- Efficiency
- Boxes, AABBs & Groupings
- To Rasterization
- 3D Engine Overview
- Rasterization





## Rasterization

tice (depth ( 1935

= = inside / 1 tt = nt / nc, dde 552t = 1.0f = nnt 5, N ); 8)

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

= diffuse; = true;

-:fl + refr)) && (depth K MADIII

), N ); ~efl \* E \* diff = true;

#### WXDEPTH)

survive = SurvivalProbabile estimation - doing it pro if; radiance = SampleLight( %rand e.x + radiance.y + radiance.z)

w = true; ot brdfPdf = EvaluateDiffuse( L, N ) ot3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely follow: /ive)

; t3 brdf = SampleDiffuse( diffuse, N, r1, r2, 48, 4; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true;



## Rasterization

tica k (depth < 182

= = inside / 1 tt = nt / nc, dde -552t = 1.0f = nnt 5, N ); 8)

st a = nt - nc, b = mt + s st Tr = 1 - (R0 + (1 Tr) R = (D \* nnt - N \*

= diffuse; = true;

-:fl + refr)) && (depth & HADIII

D, N ); refl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbabi estimation - doing it pr if; radiance = SampleLight( &rand e.x + radiance.y + radiance.z)

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

andom walk - done properly, closely follow: /ive)

; tt3 brdf = SampleDiffuse( diffuse, N, F1, F2, GR, pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## Primary Rays

## *Ray tracing versus Rasterization*

## Rasterization:

- 1. Transform primitive into camera space
- 2. Project vertices into 2D screen space
- 3. Determine which pixels are affected
- 4. Use z-buffer to sort (pixels of) primitives
- 5. Clip against screen boundaries





## Rasterization

fice ⊾ (depth ⊂ 1000

= inside / 1 it = nt / nc, ddo ss2t = 1.0f - not 3, N ); 3)

at a = nt - nc, b = nt - ncat Tr = 1 - (R0 + (1 - 1))Tr) R = (0 \* nnt - N + 1)

= diffuse; = true;

-:fl + refr)) && (depth k HAADIIII

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Purple st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* Pulple

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, SS pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## Shadow Rays

The rasterization pipeline renders triangles one at a time.

- Shading calculations remain the same
- But determining light visibility is non-trivial.

Rasterization does not have access to *global data*.



## Rasterization

#### tice (depth is NASS

: = inside / l it = nt / nc, dde os2t = 1.0f - nnt -), N ); 8)

st a = nt - nc, b - nt - --st Tr = 1 - (R0 + (1 - --fr) R = (D \* nnt - N \* ----

= diffuse; = true;

-:fl + refr)) && (depth & HADIII

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it property if; radiance = SampleLight( &rand, I, I, I, e.x + radiance.y + radiance.z) = 0)

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

andom walk - done properly, closely following: /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, Lord pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## Spaces

Ray tracing typically happens in a single 3D coordinate system.

In rasterization, we use many coordinate systems:

- Camera space
- Clip space
- 2D screen space
- Model space
- Tangent space

We need efficient tools to get from one space to another. We will make extensive use of matrices to do this.



## Rasterization

tice ≰ (depth < 10.5

= inside / 1 it = nt / nc, dde os2t = 1.0f - nnt -D, N ); D)

at a = nt - nc, b - nt - at Tr = 1 - (R0 - (1 - - - -Fr) R = (D \* nnt - N - - - -

= diffuse; = true;

-:fl + refr)) && (depth & HADDIN

D, N ); refl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability difference estimation - doing it properly if; adiance = SampleLight( %rand, I, Market e.x + radiance.y + radiance.z) > 0) %

v = true;

at brdfPdf = EvaluateDiffuse( L, N.) Pauro bit st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* ((weight \* cosThetaOut) / directPdf) \* (Paulo E \* (weight \* cosThetaOut) / directPdf) \* (Paulo E \* (weight \* cosThetaOut) / directPdf) \* (Paulo E \* (weight \* cosThetaOut) / directPdf) \* (Paulo E \* (weight \* cosThetaOut) / directPdf) \* (Paulo E \* (weight \* cosThetaOut) / directPdf) \* (weight \* cosThetaOut) / directPdf) \* (weight \* cosThetaOut) \* (weight \* cosTheta

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, brdf prvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true;

## Common Concepts

Many things remain the same:

- Normal interpolation
- Shading
- Texture mapping
- The camera
- Boxes.



tica ≰ (depth < PAx

= inside / 1 it = nt / nc, dda os2t = 1.0f - nnt 0, N(); 3)

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

= diffuse; = true;

-: efl + refr)) && (depth k HANDIIII

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; adiance = SampleLight( &rand, I, I, e.x + radiance.y + radiance.z) > 0) ##

v = true; t brdfPdf = EvaluateDiffuse( L, N ) Promote st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, brd pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## Today's Agenda:

- Efficiency
- Boxes, AABBs & Groupings
- To Rasterization
- 3D Engine Overview
- Textures





# 3D Engine

at a = nt - nc

), N ); efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( diff. adiance = SampleLight( &rand, I. LL e.x + radiance.y + radiance.z) > 0) []

v = true; st brdfPdf = EvaluateDiffuse( L, N ) st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ) at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely follo vive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, LR, st rvive; pdf; i = E \* brdf \* (dot( N, R ) / pdf); sion = true:

Topics covered so far:

### Basics:

- Rasters
- Vectors
- Color representation

## Ray tracing:

- Light transport
- Camera setup
- Textures

### Shading:

- N dot L
- Distance attenuation
- Pure specular







# 3D Engine

- ic) € (depth < 10.55
- = = inside / : it = nt / nc, d/m ss2t = 1.0f - nt -2, N ); 3)
- st a = nt nc, b + nt + st Tr = 1 - (R0 + (1 - 1 fr) R = (D \* nnt - N \*
- E \* diffuse; = true;
- -**:fl + refr)) && (depth ic NADD**
- ), N ); -efl \* E \* diffuse; = true;
- AXDEPTH)



- Rendering Functional overview
- 1. Transform: translating / rotating / scaling meshes
- 2. Project: calculating 2D screen positions
- 3. Rasterize: determining affected pixels
- 4. Shade: calculate color per affected pixel



pixels

Postprocessing





# 3D Engine

## Rendering – Data Overview

plane

camera

 $T_{car1}$ 

car

(turret)

dude )

T<sub>camera</sub>

(wheel)

(wheel)

world

car

(turret)

(dude)

plane1

(wheel)

(wheel)

 $T_{car2}$ 

(wheel)

(wheel)

T<sub>buggy</sub>

plane

plane2

), N ); = true;

AXDEPTH)

survive = SurvivalPr wheel adiance = SampleLight( e.x + radiance.y + radiance wheel) v = true; t brdfPdf = EvaluateDiffu st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPd

andom walk - done properly, closely fol vive)

at3 brdf = SampleDiffuse( diffuse, N, r1 urvive; pdf; i = E \* brdf \* (dot( N, R ) / pdf);

sion = true:





# 3D Engine

"Le: € (depth ( 192

= inside / 1 it = nt / nc, dde -552t = 1.0f - nnt -5, N ); 3)

at a = nt - hc, b + nt - hcat Tr = 1 - (R0 + (1 - 1))(r) R = (0 + nnt - 1)

= diffuse; = true;

efl + refr)) && (depth is HADDII

), N ); ~efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property ff; radiance = SampleLight( &rand, I e.x + radiance.y + radiance.r) = 0

v = true;

st brdfPdf = EvaluateDiffuse( L, N ) \* Paulos st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* 000

andom walk - done properly, closely following : /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, bp3 urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## Rendering – Data Overview

Objects are organized in a hierarchy: the *scenegraph*.

In this hierarchy, objects have translations and orientations relative to their parent node.

Relative translations and orientations are specified using matrices.

Mesh vertices are defined in a coordinate system known as *object space*.







# 3D Engine

fice € (depth < 182

= inside / 1 it = nt / nc, dde os2t = 1.0f - nnt D, N(); B)

st a = nt - nc, b = nt - ncst Tr = 1 - (R0 + (1 - 10))(r) R = (0 \* nnt - 10)

= diffuse; = true;

: :fl + refr)) && (depth & NADIII

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it properly if; radiance = SampleLight( %rand, I, I) e.x + radiance.y + radiance.r) = 0.000

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Pour 1 st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* 0000

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, D) pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true;

## Rendering – Data Overview

Transform takes our meshes from object space (3D) to camera space (3D).

Project takes the vertex data from camera space (3D) to screen space (2D).



# 3D Engine

tics Advantation of the

= diffuse; = true;

: :fl + refr)) && (depth & HANDI

D, N ); ~efl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it property if; radiance = SampleLight( &rand, I, I, I) e.x + radiance.y + radiance.r) = 0

w = true; ot brdfPdf = EvaluateDiffuse( L, N.) \* P at3 factor = diffuse \* INVPI; ot weight = Mis2( directPdf, brdfPdf ); ot cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf

andom walk - done properly, closely following vive)

; t3 brdf = SampleDiffuse( diffuse, N, r1, r2, BR 1997) pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true;



The screen is represented by (at least) two buffers:



depth information



# 3D Engine

= inside / : nt = nt / nc, ddo os2t = 1.0f - nnt -O, N ); 8)

st a = nt - nc, b - nt - st Tr = 1 - (R0 + (1 - - fr) R = (D - nnt - N - - - -

E = diffuse; = true;

-: :fl + refr)) && (depth k HANDI

D, N ); refl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it property if; radiance = SampleLight( &rand, I, I) e.x + radiance.y + radiance.z) = 0

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Pu st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following /ive)

; t33 brdf = SampleDiffuse( diffuse, N, F1, F2, F8, Fp); urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

Rendering – Components	
Scenegraph Culling	Lecture 10
Vertex transform pipeline Matrices to convert from one space to another Perspective	<i>Lecture 9 Lecture 11</i>
Rasterization Interpolation Clipping Depth sorting: z-buffer	<i>Lecture 14 Lecture 14</i>
Shading Light / material interaction Complex materials	P2 P3





};

# 3D Engine

tic: ⊾ (depth < NJ)

: = inside / l ht = nt / nc, dde bs2t = 1.0f - nnt -D, N ); B)

st a = nt - nc, b - nt st Tr = 1 - (80 + (1 Tr) R = (D \* nnt - N \*

E = diffuse; = true;

: :fl + refr)) && (depth < HAND

D, N ); ~efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property ff; radiance = SampleLight( &rand, I e.x + radiance.y + radiance.z) > 0)

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Pun st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following : /ive)

; st3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, D) urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## Transformations

Rendering a scene graph is done using a recursive function:

```
void SGNode::Render( mat4& M )
```

```
mat4 M' = M<sub>local</sub> * M;
mesh->Rasterize( M' );
for( int i = 0; i < childCount; i++ )
    child[i]->Render( M' );
```

Here, matrix concatenation is part of the recursive flow.



# 3D Engine

tice k (depth < 100

= inside / 1 tt = nt / nc, ddo -552t = 1.0f = nnt -5, N ); 8)

at  $a = nt - nc_{2} b - nt$ at Tr = 1 - (R0 + (1 - 1))Tr ) R = (D \* nnt - N - 1)

= diffuse; = true;

-:fl + refr)) && (depth & HADDIII

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it properly if; adiance = SampleLight( %rand, I =x + radiance.y + radiance.z) = 0

v = true; t brdfPdf = EvaluateDiffuse( L, N ) = Pour st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* 000

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, D) pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

Transformations

To transform meshes to world space, we call SGNode::Render with an identity matrix.

To transform meshes to camera space, we call it with the *inverse* transform of the camera.

Remember: the world revolves around the viewer; instead of turning the viewer, we turn the world in the opposite direction.

```
void SGNode::Render( mat4& M )
{
    mat4 M' = M<sub>local</sub> * M;
    mesh->Rasterize( M' );
    for( int i = 0; i < childCount; i++ )
        child[i]->Render( M' );
};
```



# 3D Engine

tice ⊾ (depth ⊂ NAS

= = inside / 1 it = nt / nc, dde ss2t = 1.0f = nnt 3, N ); 3)

at a = nt - nc, b = nt - ncat Tr = 1 - (R0 + (1 - 1))Tr) R = (0 + nnt - N - 1)

= diffuse; = true;

: :fl + refr)) && (depth k HANDIIII

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property ff; radiance = SampleLight( %rand, I e.x + radiance.y + radiance.z) > 0)

v = true; at brdfPdf = EvaluateDiffuse( L, N ) = Pour st3 factor = diffuse = INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) = 000

andom walk - done properly, closely following : /ive)

; st3 brdf = SampleDiffuse( diffuse, N, r1, r2, N, sta urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true;

## After projection

The output of the projection stage is a stream of vertices for which we know 2D screen positions.

The vertex stream must be combined with connectivity data to form triangles.

'Triangles' on a raster consist of a collection of pixels, called *fragments*.



pixels



tica ≰ (depth < PAx

= inside / 1 it = nt / nc, dda os2t = 1.0f - nnt 0, N(); 3)

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

= diffuse; = true;

-: efl + refr)) && (depth k HANDIIII

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; adiance = SampleLight( &rand, I, I, e.x + radiance.y + radiance.z) > 0) ##

v = true; t brdfPdf = EvaluateDiffuse( L, N ) Promote st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, brd pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## Today's Agenda:

- Efficiency
- Boxes, AABBs & Groupings
- To Rasterization
- 3D Engine Overview
- Textures





tice (depth c ruo

z = inside / L it = nt / nc, dd os2t = 1.0f 0, N ); 3)

at a = nt - nc, b - nt at Tr = 1 - (R0 + (1 - 1 Tr) R = (0 \* nnt - N \*

= diffuse = true;

-: :fl + refr)) && (depth is HADDEE

D, N ); refl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property if; adiance = SampleLight( &rand, I .x + radiance.y + radiance.z) 0

v = true; at brdfPdf = EvaluateDiffuse( L, N ) Process st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, brd pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

# **INFOGR – Computer Graphics**

Jacco Bikker & Debabrata Panja - April-July 2017

## END OF lecture 7: "Accelerate"

# Next lecture: "OpenGL"

