デモシーンへようこそ

4KBで映像作品を作る技術、およびゲーム開発への応用

TokyoDemoFest Organizers:
Seiya Ishibashi / Julien Guertault

はじめに

- CEDEC 2016 のセッションの続きになりますが、前回を見ていなくても大丈夫な構成になっています
- デモシーンの技術やノウハウをゲーム制作に応用してみよう、というのが今回の趣旨になります
- 辛口セッションなのでそれなりの前提知識を求めます
  - 最低限 deferred shading がどう機能するか知っているレベル
- スライドは公演後公開します

About Us

Seiya Ishibashi
(@i_saint)

Julien Guertault
(@Zavie)

- Tokyo Demo Fest の運営チーム
- 両者共ゲーム会社勤めだが、今回は TDF として講演

Agenda

1. 前回のおさらい
2. 64k intro 制作から学んだこと (Zavie)
3. ゲーム制作への応用 (i-saint)

Agenda

1. 前回のおさらい
2. 64k intro 制作から学んだこと (Zavie)
3. ゲーム制作への応用 (i-saint)

前回のおさらい

- 前回はデモシーンについて浅く広く紹介し、4k intro の作り方について大雑把に解説しました
  - https://onedrive.live.com/view.aspx?resid=61562C8AE88D716B!4355
- 今回のセッションはレンダリングに絞った内容になっており、Raymarching に限って前回の内容をおさらいします

前回のおさらい

- 4k intro = 4KB の exe によるデモ
- 容量の制限からポリゴンを用いない特殊なレンダリング手法がよく用いられる
- それが Distance Function & Raymarching

Distance Function & Raymarching

- 画面全体を覆う 4 角形を 1 枚出す
- あとは全てピクセルシェーダでがんばる

Distance Function & Raymarching

- モデルは Pixel Shader の中で数式で表現し、それを可視化していく
- このモデルの数式が Distance Function、可視化のプロセスが Raymarching と呼ばれる

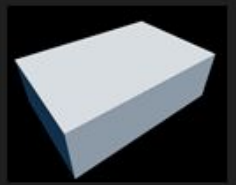
Distance Function & Raymarching

Sphere - signed - exact

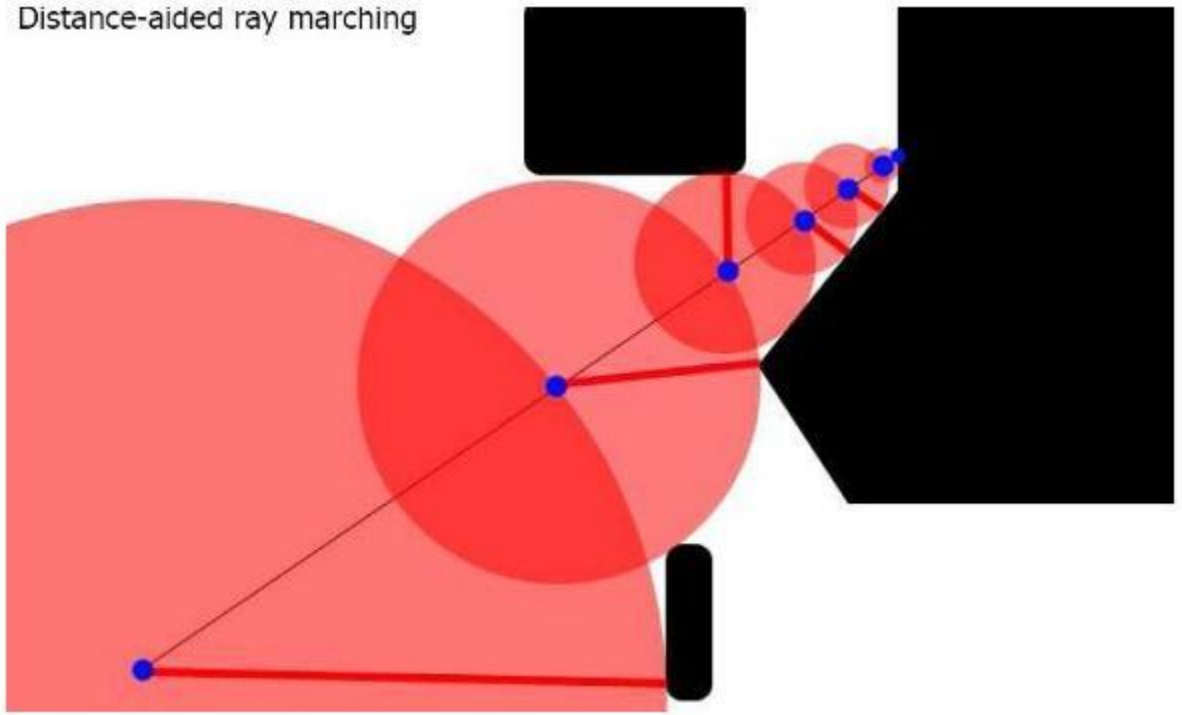
```
float sdSphere( vec3 p, float s )
{
  return length(p)-s;
}
```

Box - signed - exact

```
float sdBox( vec3 p, vec3 b )
{
  vec3 d = abs(p) - b;
  return min(max(d.x,max(d.y,d.z)),0.0) + length(max(d,0.0));
}
```



Distance Function & Raymarching



Distance Function & Raymarching

- 十分に距離が小さくなったら計算を打ち切る
  - もしくは上限マーチ回数を超えたら打ち切る
  - 完全に正確な交差点は求まらないが、必要十分
- この計算を全てのピクセルに対して行う
  - Distance Function の複雑さ * マーチ回数 に応じて負荷が増大
- 特徴的なレイの進み方から Sphere Tracing と呼ぶことも

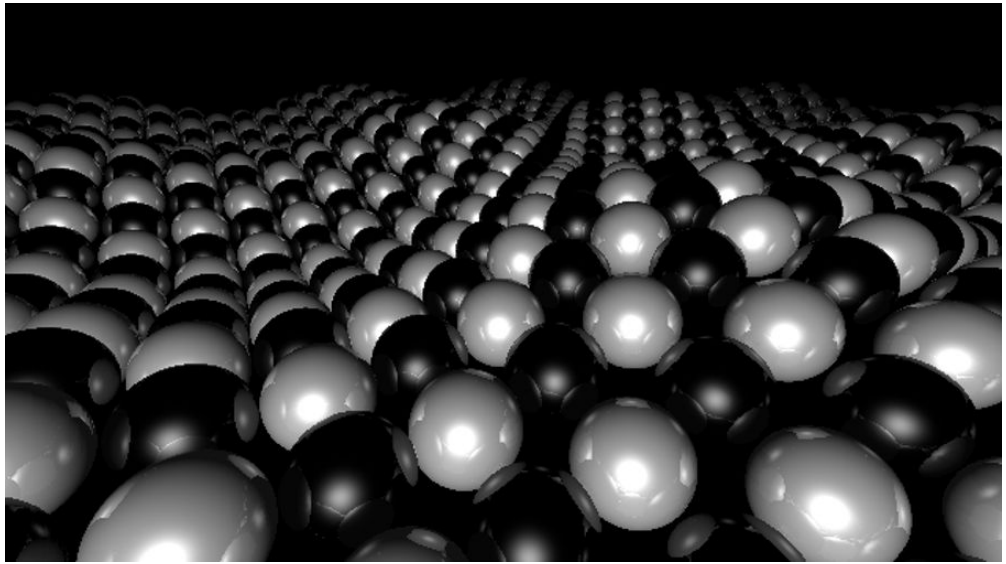
Distance Function & Raymarching

- 法線を求めることも可能
- 交差点からx,y,z軸それぞれ前後に少しずらした地点との距離を計算
- 勾配から法線を推測

```
vec3 guess_normal(vec3 p)
{
    const float d = 0.001;
    return normalize( vec3(
        distance_function(p+vec3(  d,0.0,0.0)) - distance_function(p+vec3( -d,0.0,0.0)),
        distance_function(p+vec3(0.0,  d,0.0)) - distance_function(p+vec3(0.0, -d,0.0)),
        distance_function(p+vec3(0.0,0.0,  d)) - distance_function(p+vec3(0.0,0.0, -d)) ));
}
```

Distance Function & Raymarching

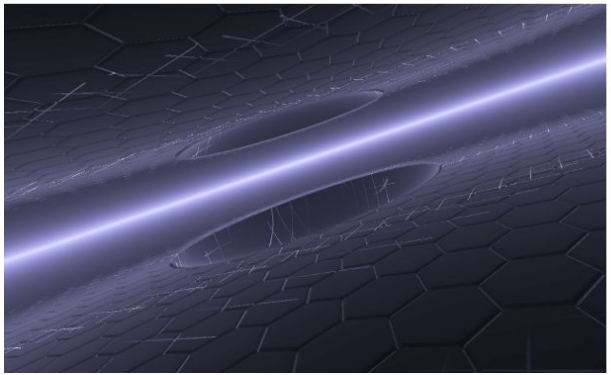
- 法線が求まるのでライティングも可能
- Ray→法線 の反射方向へ再度レイを飛ばせば反射なども表現可能



Distance Function & Raymarching

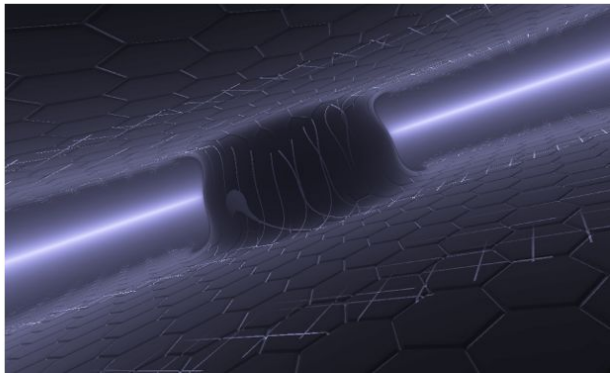
- ポリゴンでは難しいモデルの合成が簡単に実現できる

```
float subtract( float d1, float d2 )
{
    return max(-d1,d2);
}
```



```
float and( float d1, float d2 )
{
    return max(d1,d2);
}
```

```
float soft_min(float d1, float d2, float r)
{
    float e = max(r - abs(d1 - d2), 0.0);
    return min(d1, d2) - e*e*0.25 / r;
}
```

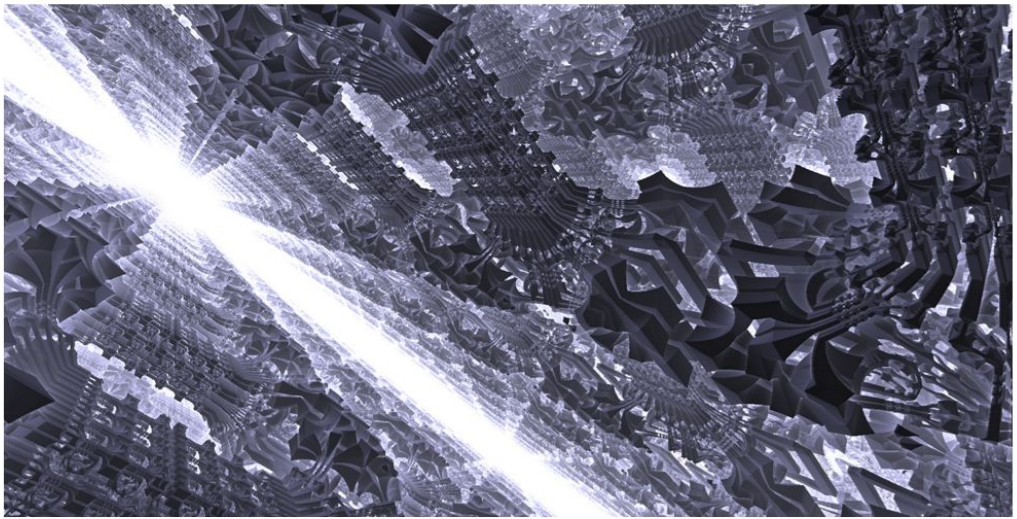


Distance Function & Raymarching

- フラクタル

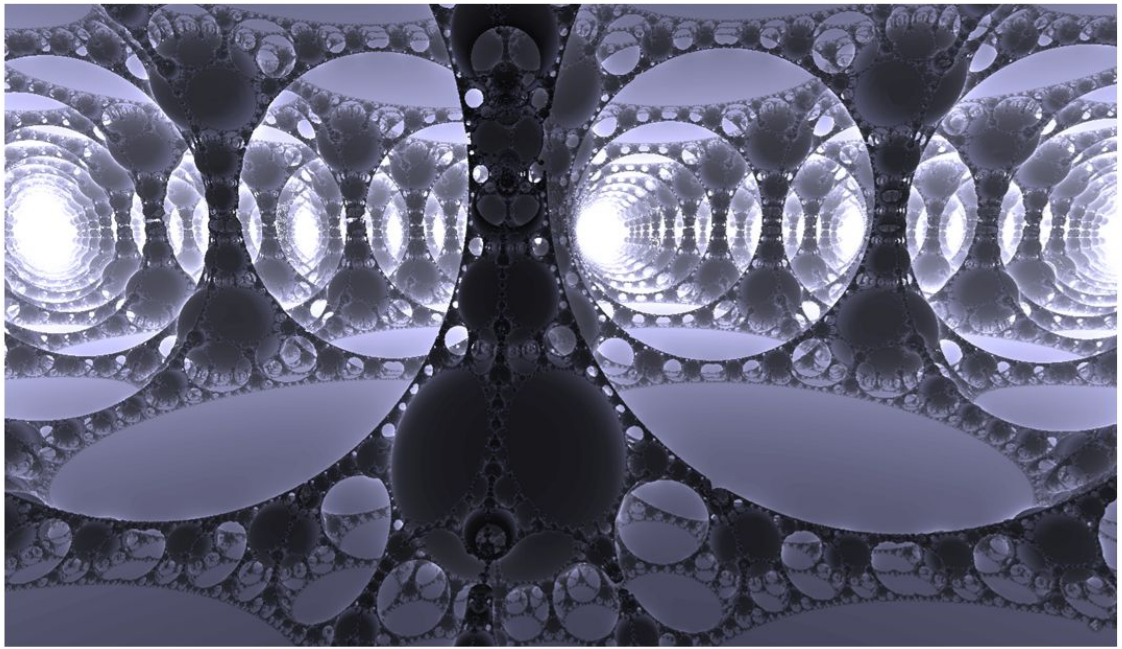
```
float tglad_formula(vec3 z0)
{
    z0 = mod(z0, 2.);

    float mr=0.25, mxr=1.0;
    vec4 scale=vec4(-3.12,-3.12,-3.12,3.12), p0=vec4(0.0,1.59,-1.0,0.0);
    vec4 z = vec4(z0,1.0);
    for (int n = 0; n < 3; n++) {
        z.xyz=clamp(z.xyz, -0.94, 0.94)*2.0-z.xyz;
        z*=scale/clamp(dot(z.xyz,z.xyz),mr,mxr)*1.;
        z+=p0;
    }
    float dS=(length(max(abs(z.xyz)-vec3(1.2,49.0,1.4),0.0))-0.06)/z.w;
    return dS;
}
```



Distance Function & Raymarching

```
float pseudo_kleinian(vec3 p)
{
    const vec3 CSize = vec3(0.92436,0.90756,0.92436);
    const float Size = 1.0;
    const vec3 C = vec3(0.0,0.0,0.0);
    float DEfactor=1.;
    const vec3 Offset = vec3(0.0,0.0,0.0);
    vec3 ap=p+1.;
    for(int i=0;i<10 ;i++){
        ap=p;
        p=2.*clamp(p, -CSize, CSize)-p;
        float r2 = dot(p,p);
        float k = max(Size/r2,1.);
        p *= k;
        DEfactor *= k;
        p += C;
    }
    float r = abs(0.5*abs(p.z-Offset.z)/DEfactor);
    return r;
}
```



Agenda

1. 前回のおさらい
2. 64k intro 制作から学んだこと (Zavie)
3. ゲーム制作への応用 (i-saint)

デモシーンについて

Professional **and** hobbyist programmer.

“What do you do on your free time?”
“Same as at work.”

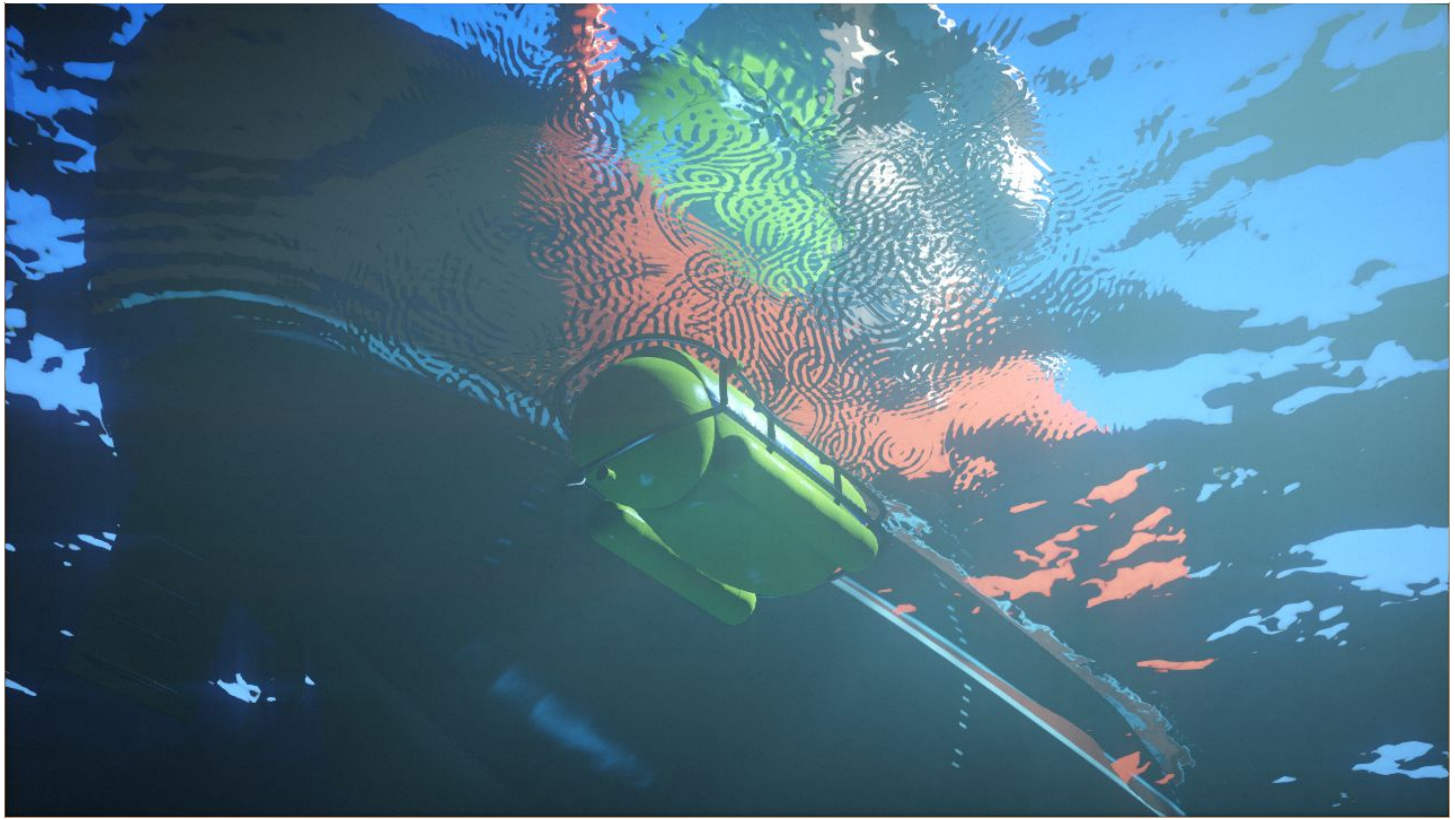
Demoscene: making demos with *Ctrl-Alt-Test.*

デモシーンについて

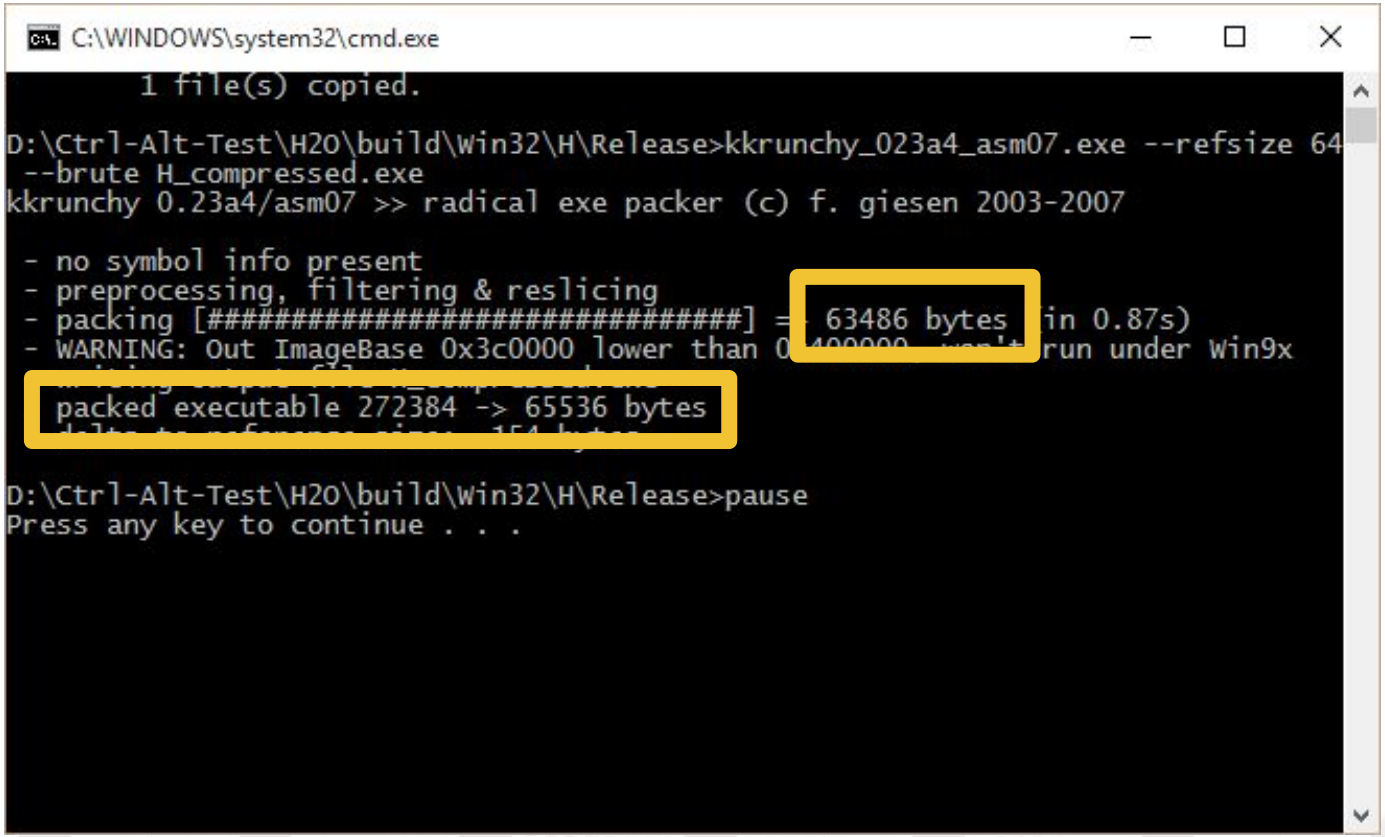
*Ctrl-Alt-Test*と言うグループ

- 64k intros (=64kBなデモ) を作ります
- プログラマー2人
- サウンド~~1人~~2人
- たまに、他の友人も参加

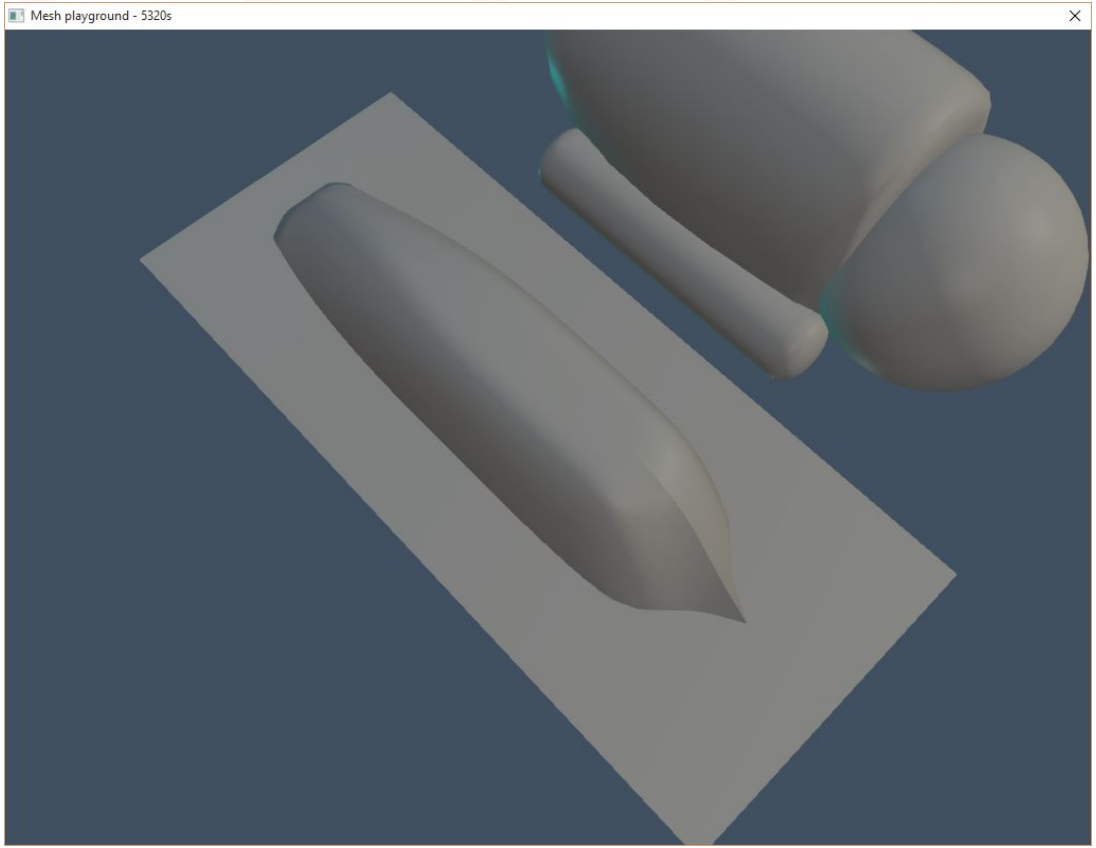
64k intro: H - Immersion (2017)



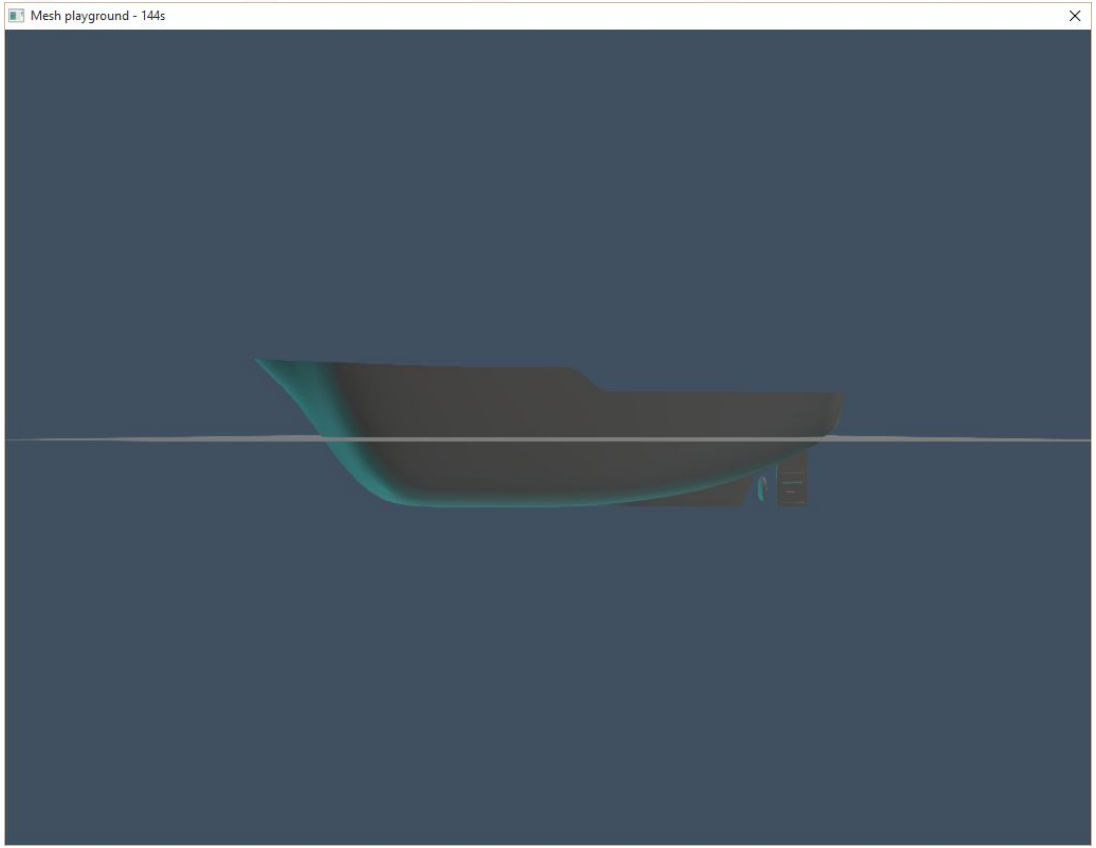
64k intro: H - Immersion (2017)



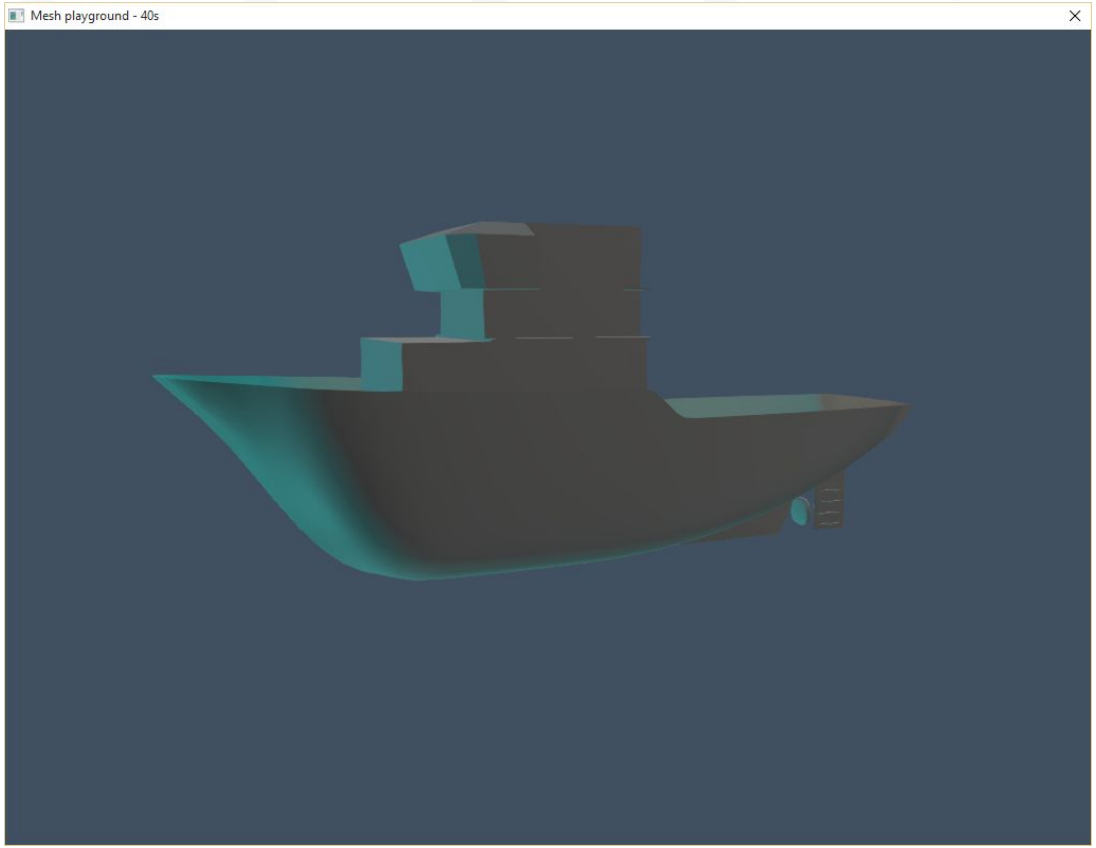
例・調査船



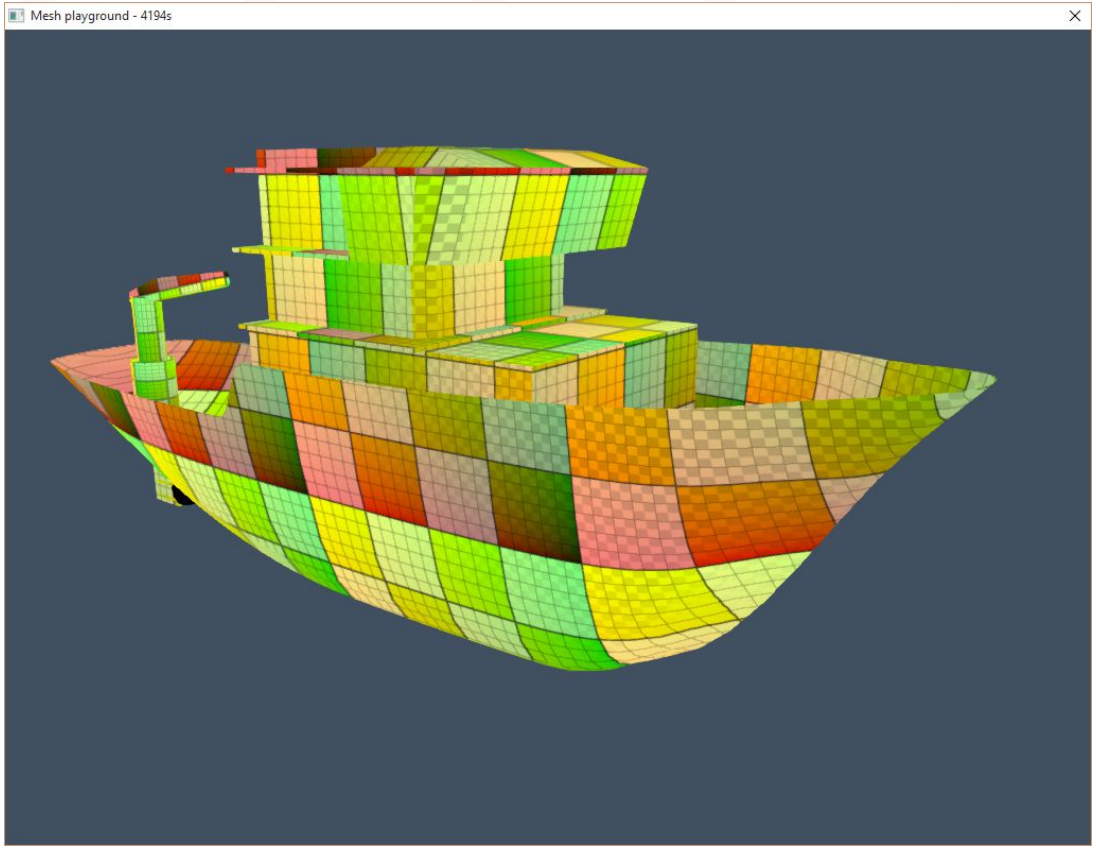
例・調査船



例・調査船



例・調査船



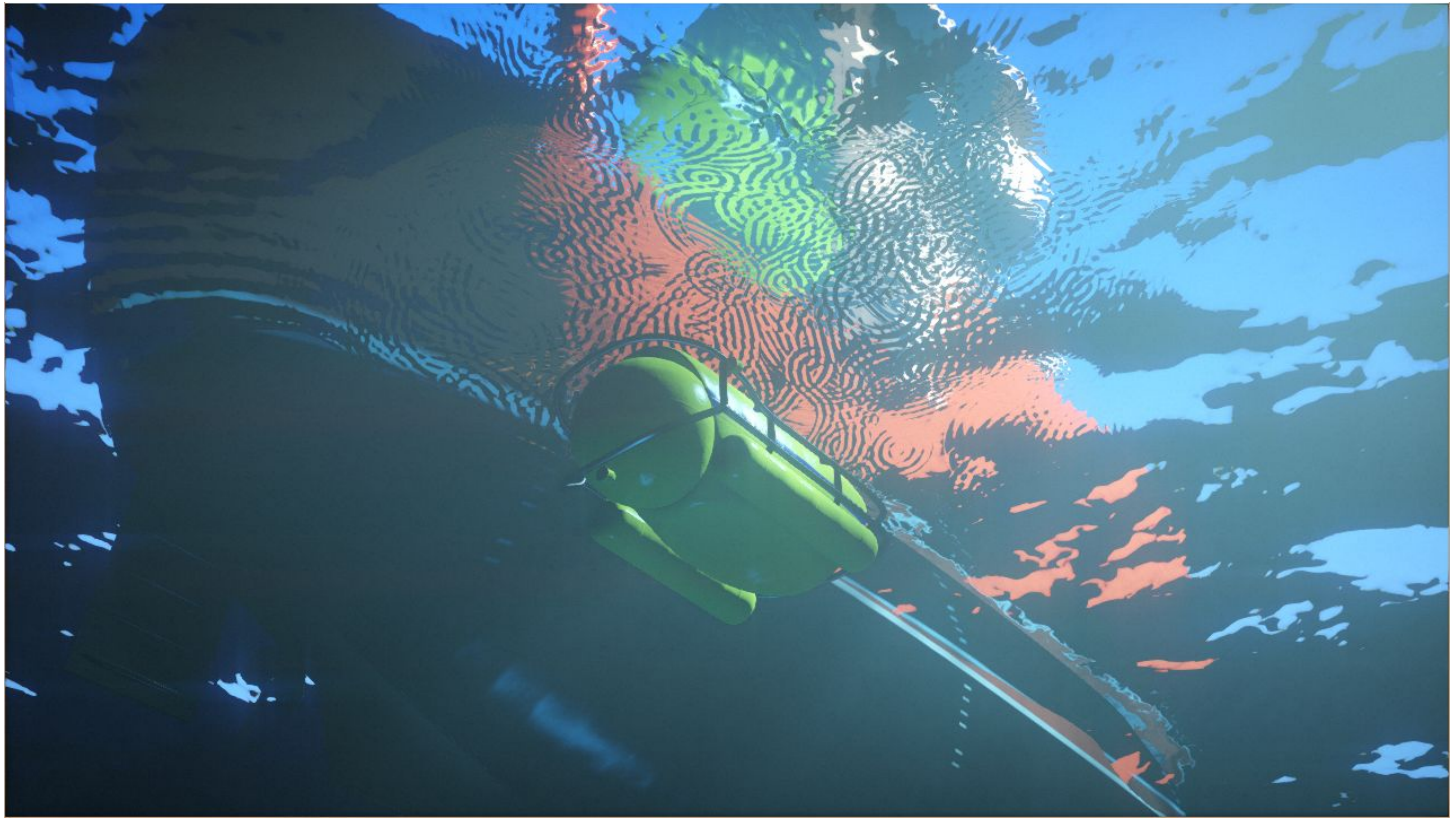
例・調査船


Mesh playground - 362s
メッシュ: ~934 バイト
テクスチャ: ~1013 バイト
misc/shared: +20~25% (?)

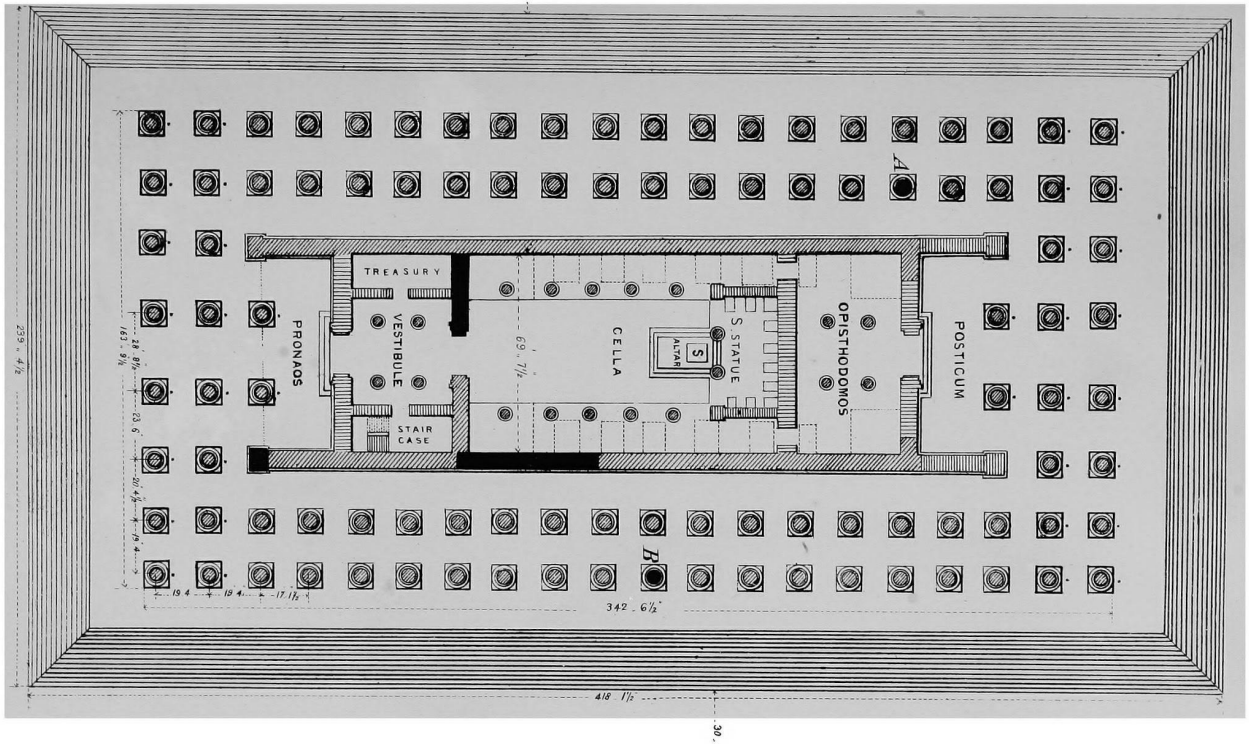
例・調査船

例・調査船

例・調査船

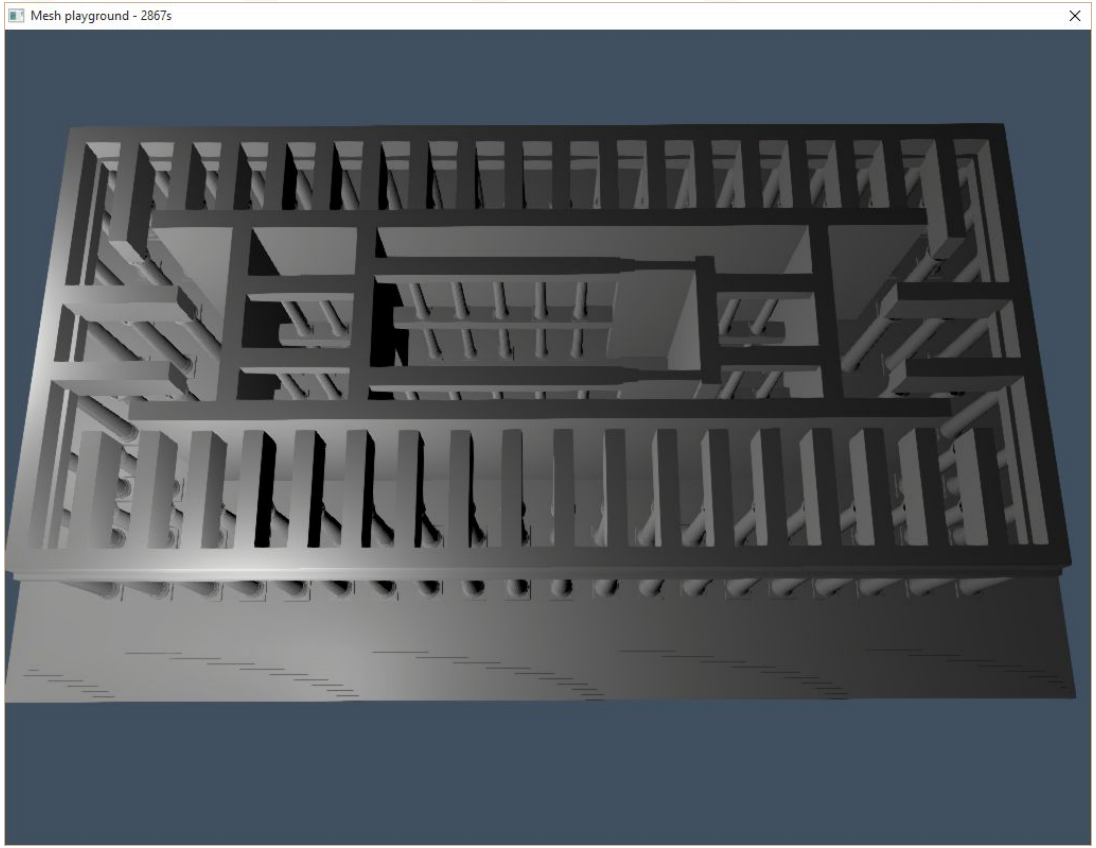


例・アルテミス神殿

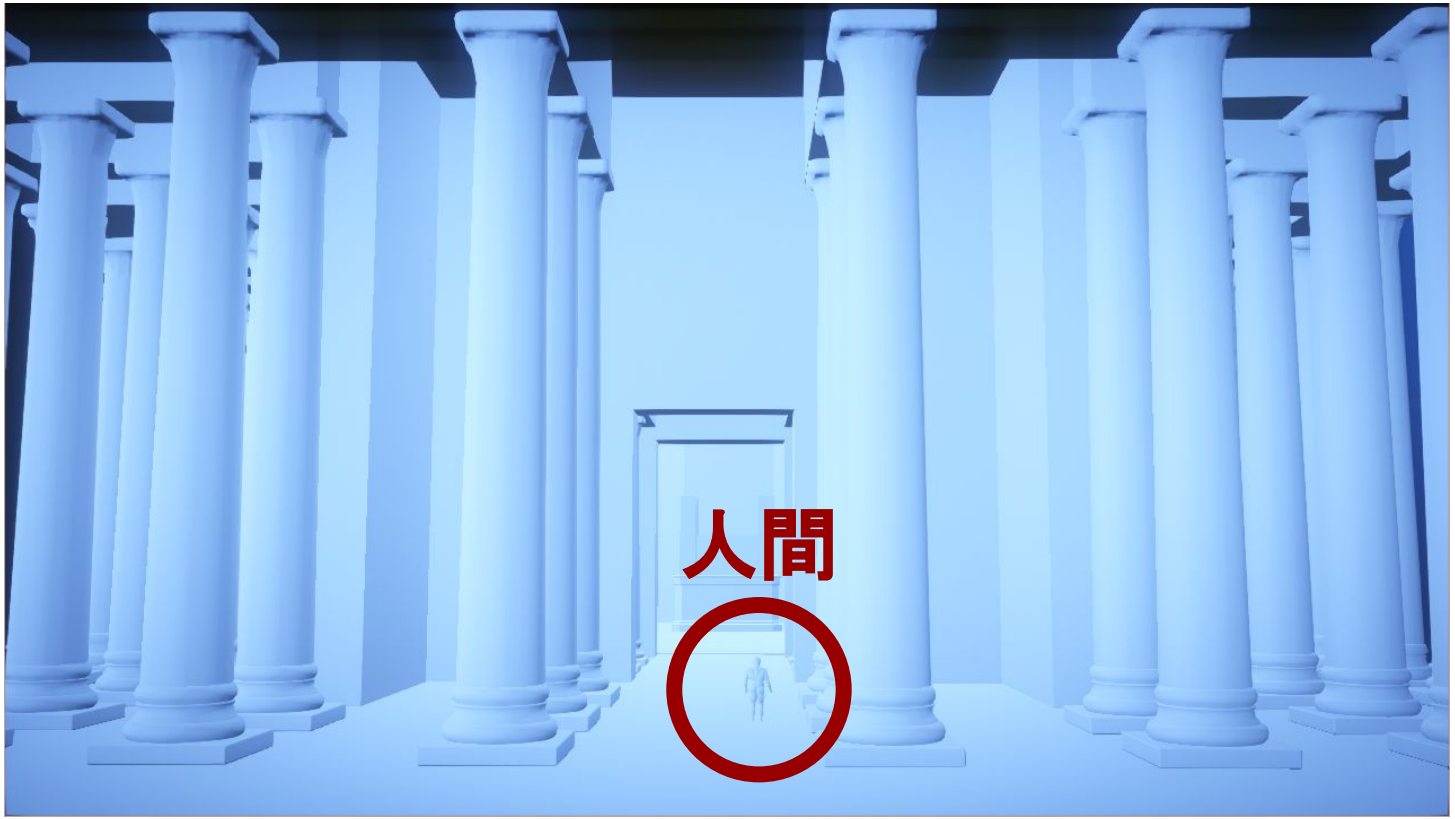


世界の七不思議のひとつに挙げられている

例・アルテミス神殿



例・アルテミス神殿



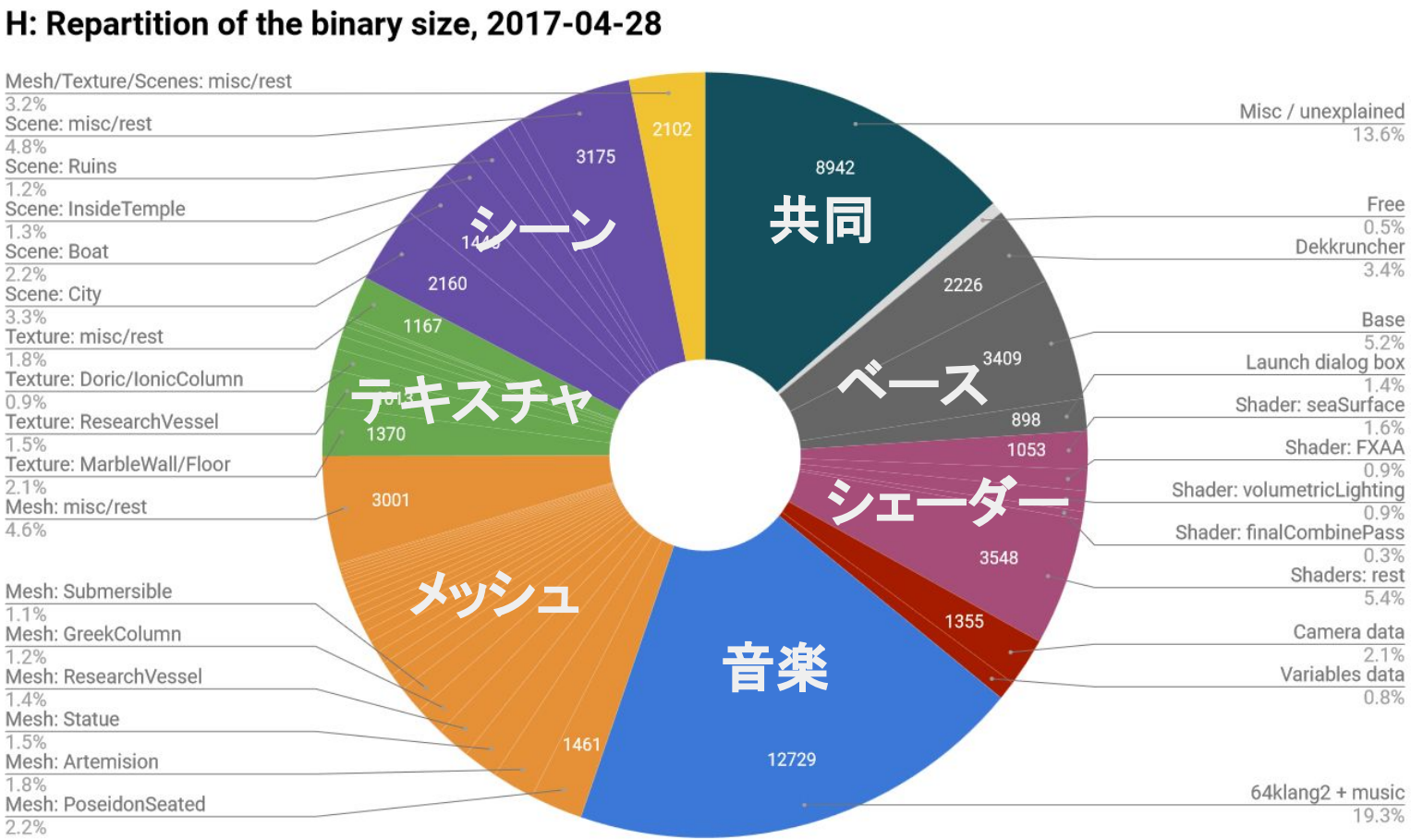
例・アルテミス神殿

例・アルテミス神殿

例・アルテミス神殿


アルテミス神殿のメッシュ: ~1174 バイト
柱のメッシュ: ~817 バイト
塑像のメッシュ: ~1461 バイト
アルテミス神殿のテクスチャ: ~1370 バイト
柱のテクスチャ: ~592 バイト
misc/shared: +20~25% (?)

サイズについて


H: Repartition of the binary size, 2017-04-28
Mesh/Texture/Scenes: misc/rest
3.2%
Scene: misc/rest
4.8%
Scene: Ruins
1.2%
Scene: InsideTemple
1.3%
Scene: Boat
2.2%
Scene: City
3.3%
Texture: misc/rest
1.8%
Texture: Doric/IonicColumn
0.9%
Texture: ResearchVessel
1.5%
Texture: MarbleWall/Floor
2.1%
Mesh: misc/rest
4.6%
Mesh: Submersible
1.1%
Mesh: GreekColumn
1.2%
Mesh: ResearchVessel
1.4%
Mesh: Statue
1.5%
Mesh: Artemision
1.8%
Mesh: PoseidonSeated
2.2%
3175
2102
8942
共同
シーン
2160
1167
テキスチャ
1370
3001
メッシュ
1461
12729
音楽
Misc / unexplained
13.6%
Free
0.5%
Dekkruncher
3.4%
2226
Base
5.2%
Launch dialog box
1.4%
Shader: seaSurface
1.6%
Shader: FXAA
0.9%
Shader: volumetricLighting
0.9%
Shader: finalCombinePass
0.3%
Shaders: rest
5.4%
3409
ベース
898
1053
シェーダー
3548
1355
Camera data
2.1%
Variables data
0.8%
64klang2 + music
19.3%

作成について

Size-codingは楽しい。それに、面白いです。

But not very useful. :-)

So let’s talk about more useful things.

作成について

*Ctrl-Alt-Test*と言うグループ

- 64k intros (=64kBなデモ) を作ります

⇒ ~~UE4, Unity, Photoshop, Maya, mp3~~ :-(

- プログラマー2人
- サウンド2人

⇒ Very **limited manpower** :-(

作成について

Reduce iteration time:

- Hot reload data!
- Hot reload shaders!
- Hot reload assets!
- Hot reload code! \o/

作成について

Hot reload code:

- Parse array at runtime; include at compile time.
- Tweakable values.

  https://github.com/joeld42/ld48jovoc/blob/master/ld17_island/tweakval.h

  https://www.gamedev.net/resources/_/technical/game-programming/tweakable-constants-r2731
- Interpreted C.

  https://github.com/zsaleeba/picoc
- Runtime recompilation.

  https://github.com/RuntimeCompiledCPlusPlus/RuntimeCompiledCPlusPlus

デバッグについて

Abuse your tools!

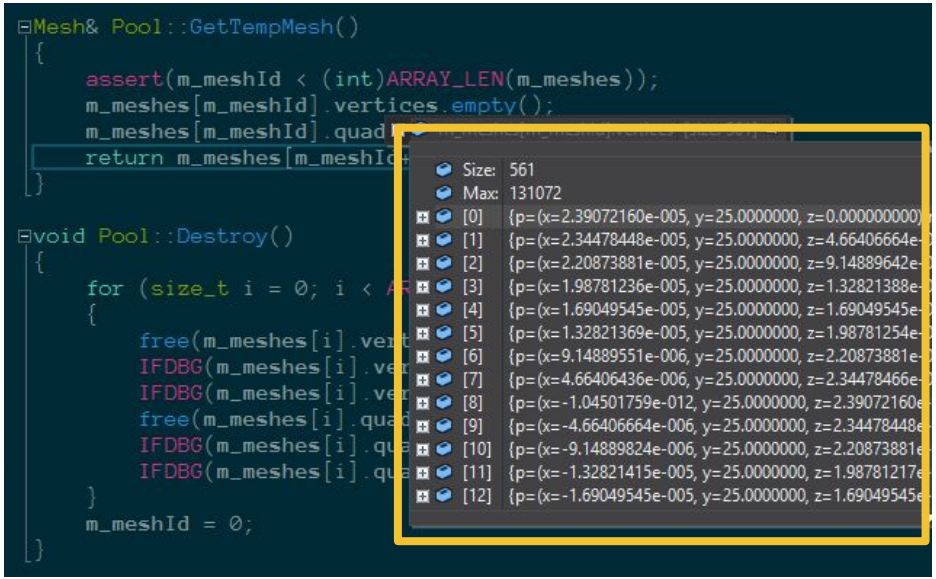
デバッグについて

Visual Studio?

- Clever syntax highlighting.
- XML comments.
- Natvis.

デバッグについて


Mesh& Pool::GetTempMesh()
{
assert(m_meshId < (int)
m_meshes[m_meshId].ver
m_meshes[m_meshId].quad
return m_meshes[m_mesh
}
#define ARRAY_LEN(array) (sizeof(array) / sizeof(array[0]))
Number of elements of a statically declared const array.
void Pool::Destroy()
{
for (size_t i = 0; i < ARRAY_LEN(m_meshes); ++i)
{
free(m_meshes[i].vertices.elt);
IFDBG(m_meshes[i].vertices.max_size = 0);
IFDBG(m_meshes[i].vertices.elt = 0);
free(m_meshes[i].quads.elt);
IFDBG(m_meshes[i].quads.max_size = 0);
IFDBG(m_meshes[i].quads.elt = 0);
}
m_meshId = 0;
}


Mesh& Pool::GetTempMesh()
{
assert(m_meshId < (int)ARRAY_LEN(m_meshes));
m_meshes[m_meshId].vertices.empty();
m_meshes[m_meshId].quad
return m_meshes[m_meshId
}
void Pool::Destroy()
{
for (size_t i = 0; i < A
{
free(m_meshes[i].ver
IFDBG(m_meshes[i].ve
IFDBG(m_meshes[i].ve
free(m_meshes[i].qua
IFDBG(m_meshes[i].qu
IFDBG(m_meshes[i].qu
}
m_meshId = 0;
}
Size: 561
Max: 131072
[0] {p=(x=2.39072160e-005, y=25.0000000, z=0.000000000)
[1] {p=(x=2.34478448e-005, y=25.0000000, z=4.66406664e
[2] {p=(x=2.20873881e-005, y=25.0000000, z=9.14889642e
[3] {p=(x=1.98781236e-005, y=25.0000000, z=1.32821388e
[4] {p=(x=1.69049545e-005, y=25.0000000, z=1.69049545e
[5] {p=(x=1.32821369e-005, y=25.0000000, z=1.98781254e
[6] {p=(x=9.14889551e-006, y=25.0000000, z=2.20873881e
[7] {p=(x=4.66406436e-006, y=25.0000000, z=2.34478466e
[8] {p=(x=-1.04501759e-012, y=25.0000000, z=2.39072160e
[9] {p=(x=-4.66406664e-006, y=25.0000000, z=2.34478448e
[10] {p=(x=-9.14889824e-006, y=25.0000000, z=2.20873881e
[11] {p=(x=-1.32821415e-005, y=25.0000000, z=1.98781217e
[12] {p=(x=-1.69049545e-005, y=25.0000000, z=1.69049545e

デバッグについて

Have an error message?

- Output log to the IDE output.
  win32: `OutputDebugString()`
- Include file location.
- Format it like a compilation error.

→ Will appear in IDE console; double click will jump to location.

デバッグについて

Have an abstraction?

Graphic API abstraction class?

Write a debug implementation.

→ Insert a debug layer.

デモ

H - Immersion

64k intro, 2017

Agenda

1. 前回のおさらい
2. 64k intro 制作から学んだこと (Zavie)
3. ゲーム制作への応用 (i-saint)

ゲーム制作への応用

Pros:

- 比較的少ない労力で複雑なモデルを出せる
- ポリゴンベースでは難しい表現ができる

Cons:

- シーン全てを Distance Function で表現するのは無理がある
- 編集や制御が大変
- 重い

ポリゴンモデルとの混在

- シーン全てを Distance Function で表現するのは無理がある
  - ポリゴンモデルと Distance Function を混在できるようにしたい
- Deferred Shading であればこれは容易に可能

ポリゴンモデルとの混在

- Deferred Shading の場合ライティングは G-Buffer に対して行われる
- G-Buffer さえ生成できれば一貫したライティングやポストエフェクト処理を実現できる
- Raymarching で G-Buffer を生成すればポリゴンモデルと完璧な混在が可能

G-Buffer 生成

- Raymarching とポリゴンモデルで座標系を一致させる必要がある
- Raymarching に必要なパラメータは ViewProjection 行列から抽出可能

```
1  // 左手座標系を想定
2  // float4x4 view, proj;
3
4  float3 forward = -view[2].xyz;
5  float3 up      = view[1].xyz;
6  float3 right   = view[0].xyz;
7  float  focal_length = abs(proj[1][1]);
8
9  float3x3 view33 = float3x3(view[0].xyz, view[1].xyz, view[2].xyz);
10 float3 rcam_pos= float3(view[0].w, view[1].w, view[2].w);
11 float3 cam_pos = mul(transpose(view33), -rcam_pos);
```

G-Buffer 生成

- Ray の位置に ViewProjection 行列を掛けて Depth を算出
- Pixel Shader の Depth 出力機能で出力
  - HLSL だと SV_Depth
  - プラットフォームよって範囲が異なる可能性があるので注意
    - 0.0~1.0 or -1.0~1.0

```
float4 ray_vp = mul(view_proj, float4(ray_pos, 1.0));
float depth = ray_vp.z / ray_vp.w;
```

G-Buffer 生成

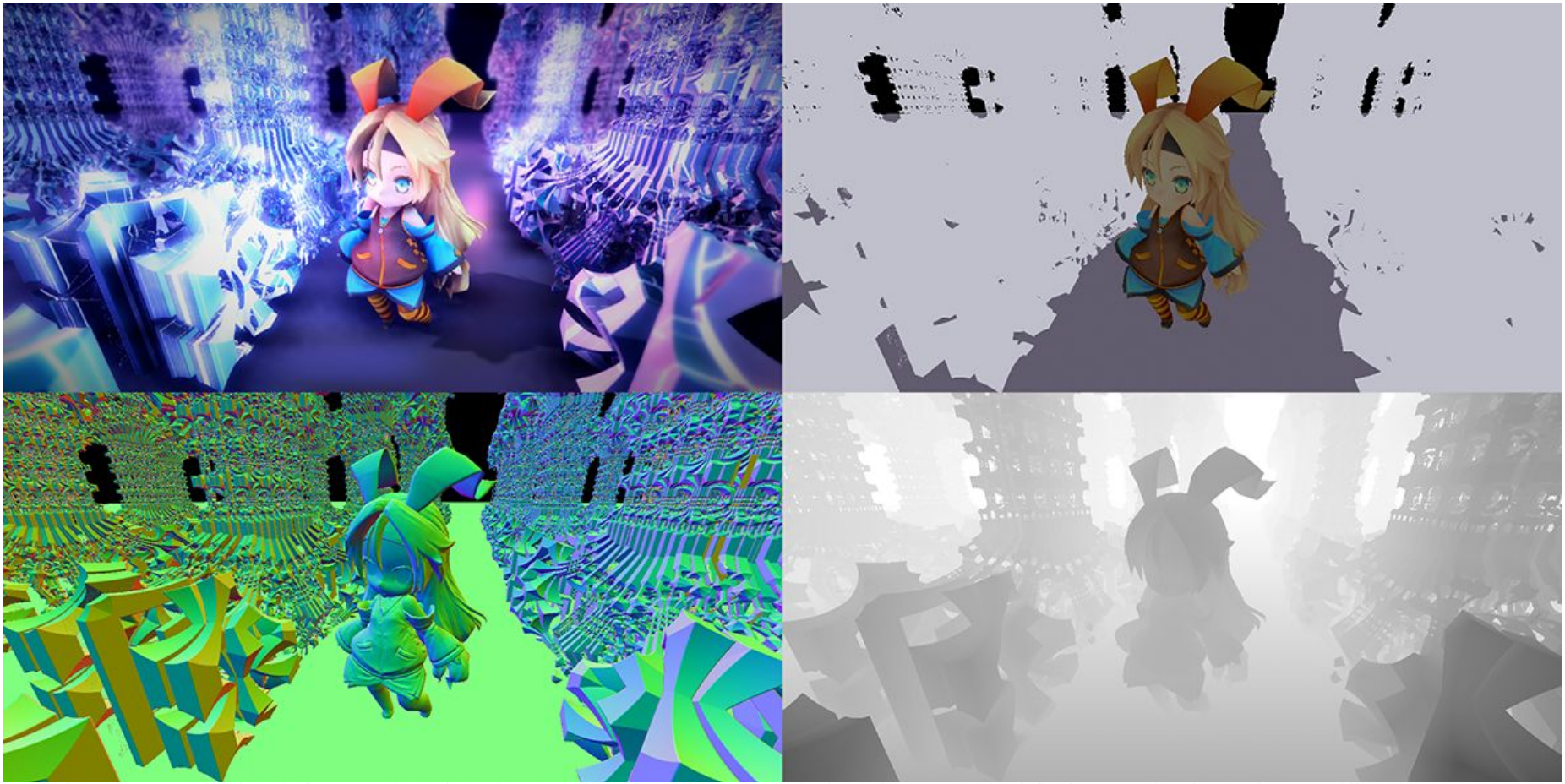
- Normal
  - 通常通り勾配から求める

```
vec3 guess_normal(vec3 p)
{
    const float d = 0.001;
    return normalize( vec3(
        distance_function(p+vec3(  d,0.0,0.0)) - distance_function(p+vec3( -d,0.0,0.0)),
        distance_function(p+vec3(0.0,  d,0.0)) - distance_function(p+vec3(0.0, -d,0.0)),
        distance_function(p+vec3(0.0,0.0,  d)) - distance_function(p+vec3(0.0,0.0, -d)) ));
}
```

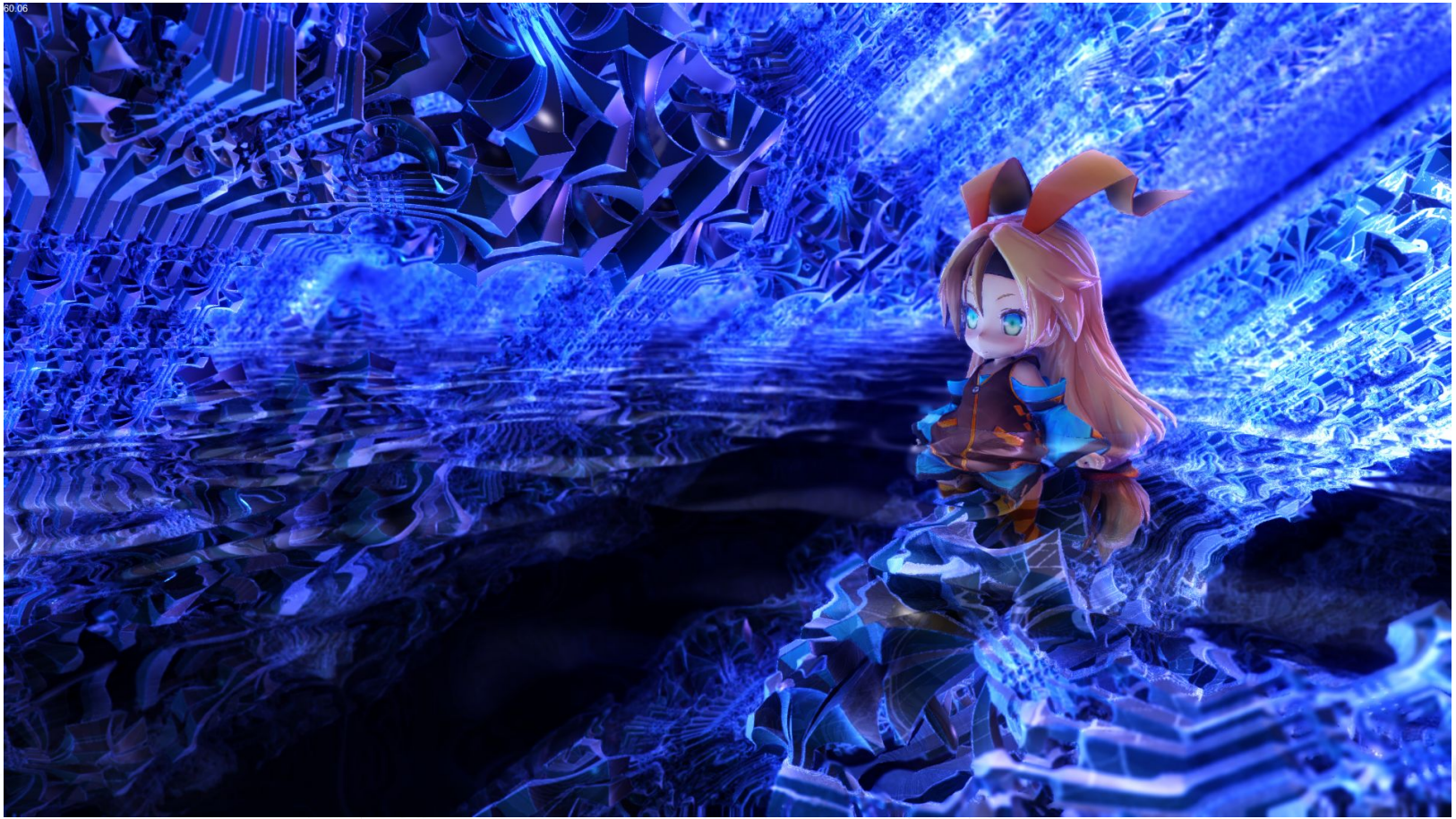
G-Buffer 生成

- Albedo, Emission など
  - 特に決まった方法はない
  - 出したい絵に応じて臨機応変に

G-Buffer 生成



G-Buffer 生成



Forward Shading の場合

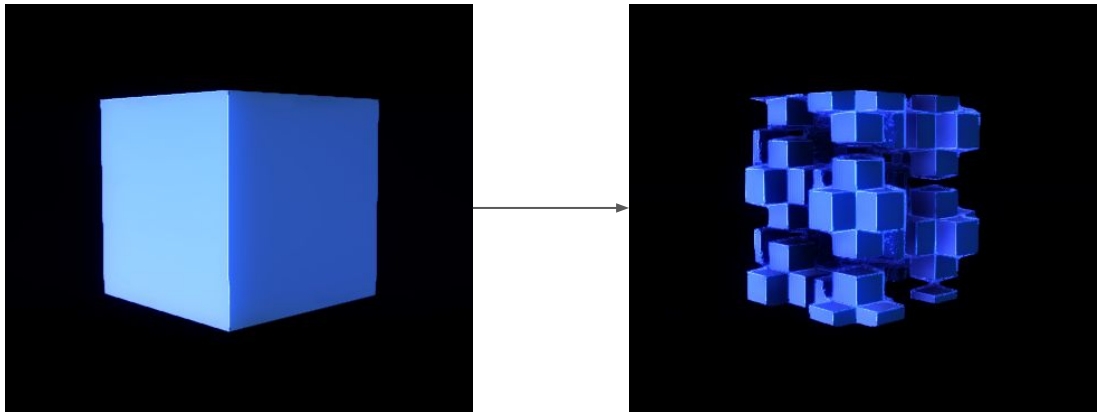
- Raymarching とライティングを 1 Pass で行えば混在可能
- Depth / Normal Buffer などを要求される場合、専用のバッファを用意
    - Ray が進んだ距離を記憶して各 Pass で使い回す

Object Space Raymarching

- Intro の場合 Fullscreen Quad 1 枚で全てをレンダリングする
  - 1 つの Distance Function で全オブジェクトを表現
- ゲームの場合この制約は不要
  - ゲーム用にシーンの構成を改善する余地がある

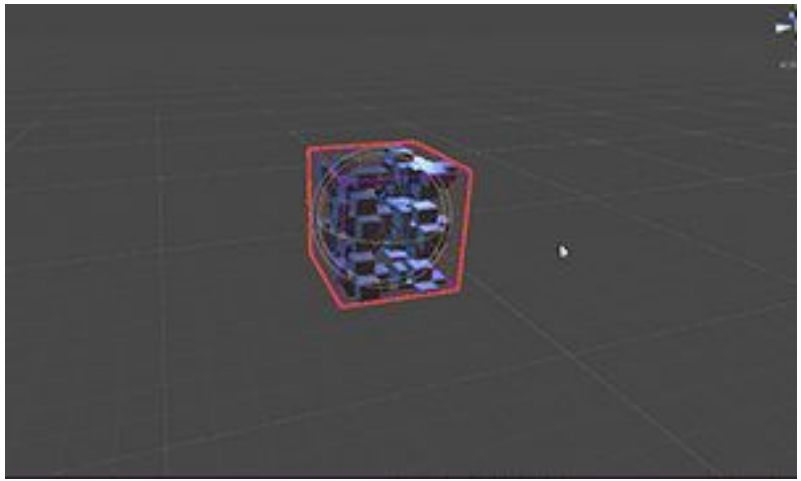
Object Space Raymarching

- Distance Function をパーツ毎に独立したシェーダに分解
- Bounding Volume となる単純なポリゴンモデルをシーンに配置
  - Box, Sphere など
- ポリゴンモデルの表面から Raymarching



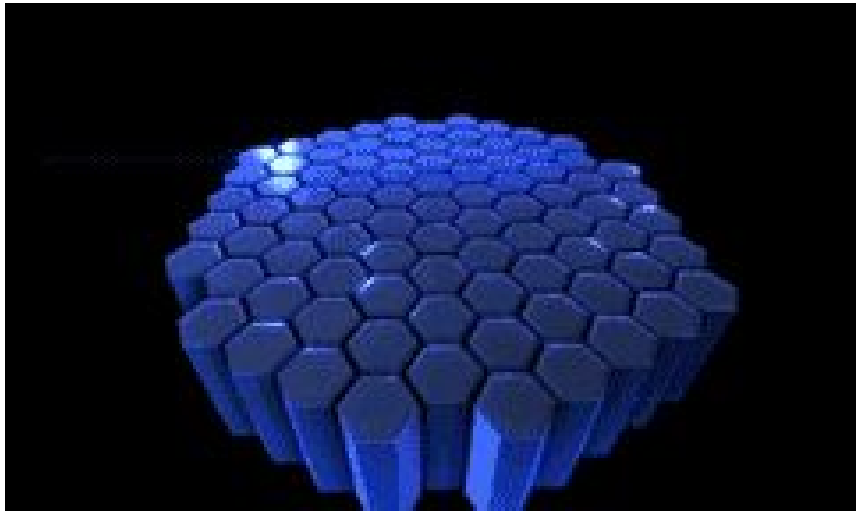
Object Space Raymarching

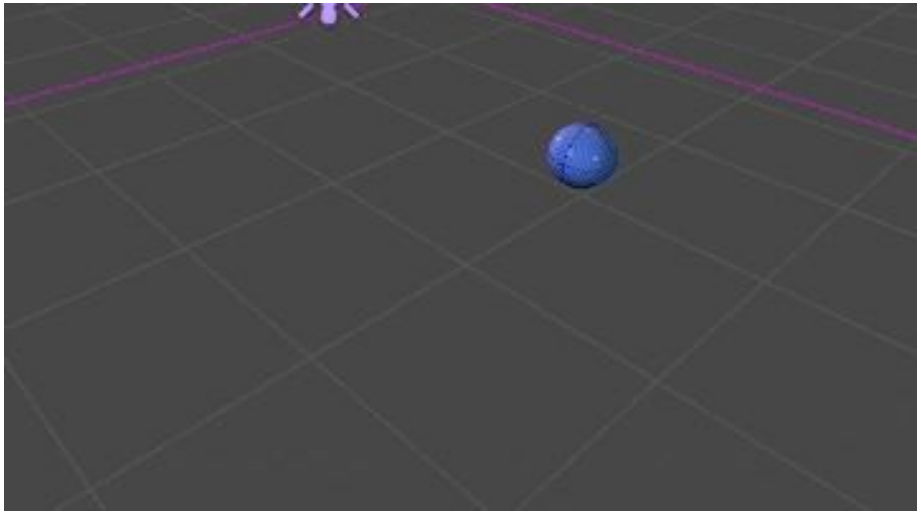
- オブジェクトの Transform を反映させる
  - Ray にオブジェクトの Transform の逆行列を掛ける
- ゲームエンジン上で Distance Function モデルの配置が可能になる



Object Space Raymarching

- パラメータによる形状の変化
  - エディタ上のパラメータ調整でバリエーションを出す





Object Space Raymarching

- 作例

Bounding Volume

Result

Raymarching 高速化

- Raymarching の負荷 3 要因
  - Distance Function の複雑さ
  - Raymarching の Step 数
  - 描画 Pixel 数
- この 3 つに比例して負荷が増大

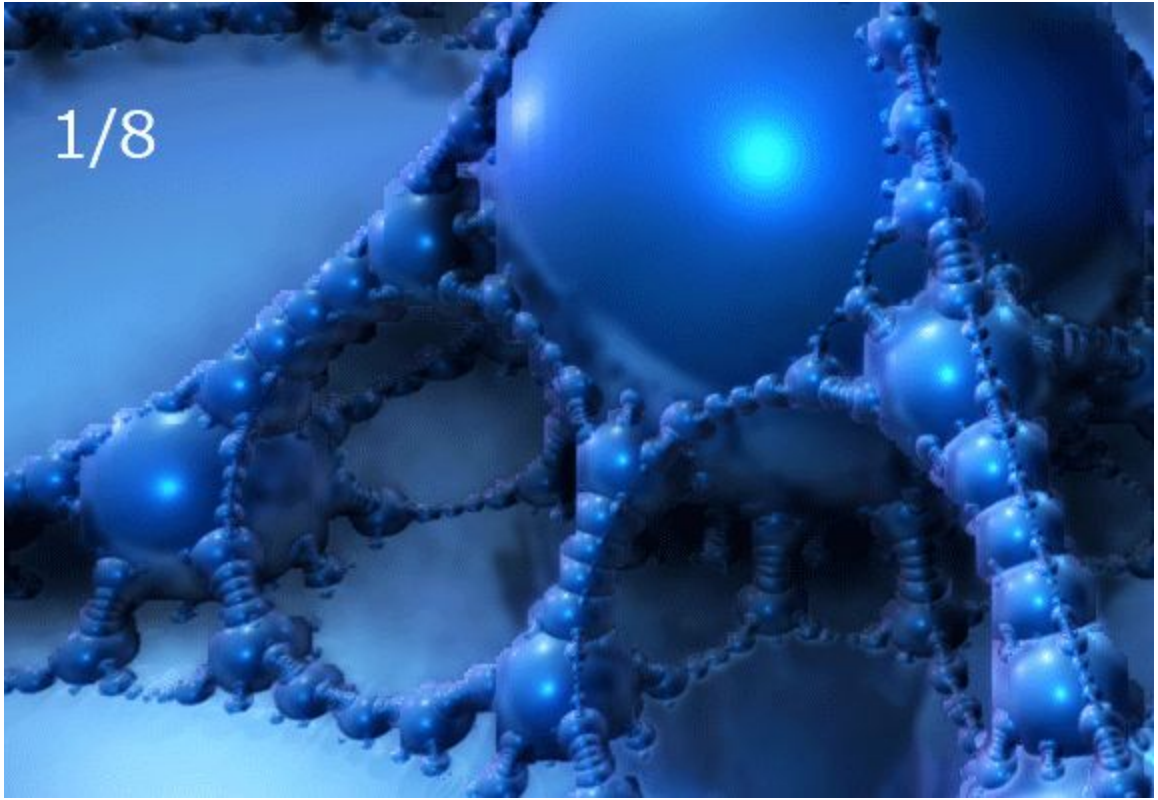
Raymarching 高速化

- Raymarching の負荷 3 要因
  - Distance Function の複雑さ
  - Raymarching の Step 数
  - 描画 Pixel 数
- この 2 つを最適化する方法を考える
  - これらは一度実装すれば Distance Function が変わっても適用可

Hierarchical Raymarching

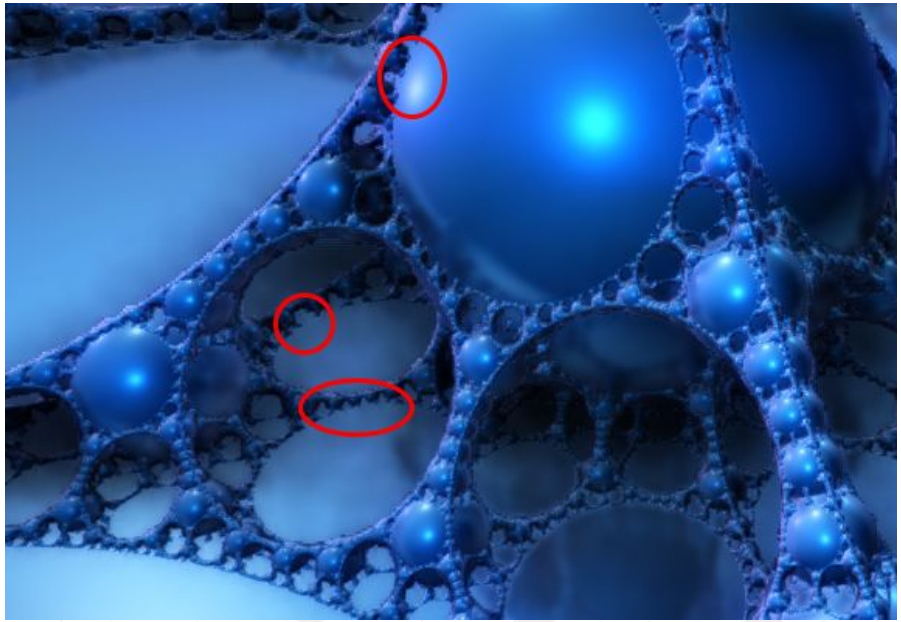
- 低解像度のバッファから段階的に Raymarching することでトータル Step 数を減らす
  - 低解像度のバッファで Step 数多めに Raymarching
  - 高解像度のバッファで、低解像度の結果を引き継いで Step 数少なめに Raymarching
  - 上記手順を何段階か繰り返す

Hierarchical Raymarching



Hierarchical Raymarching

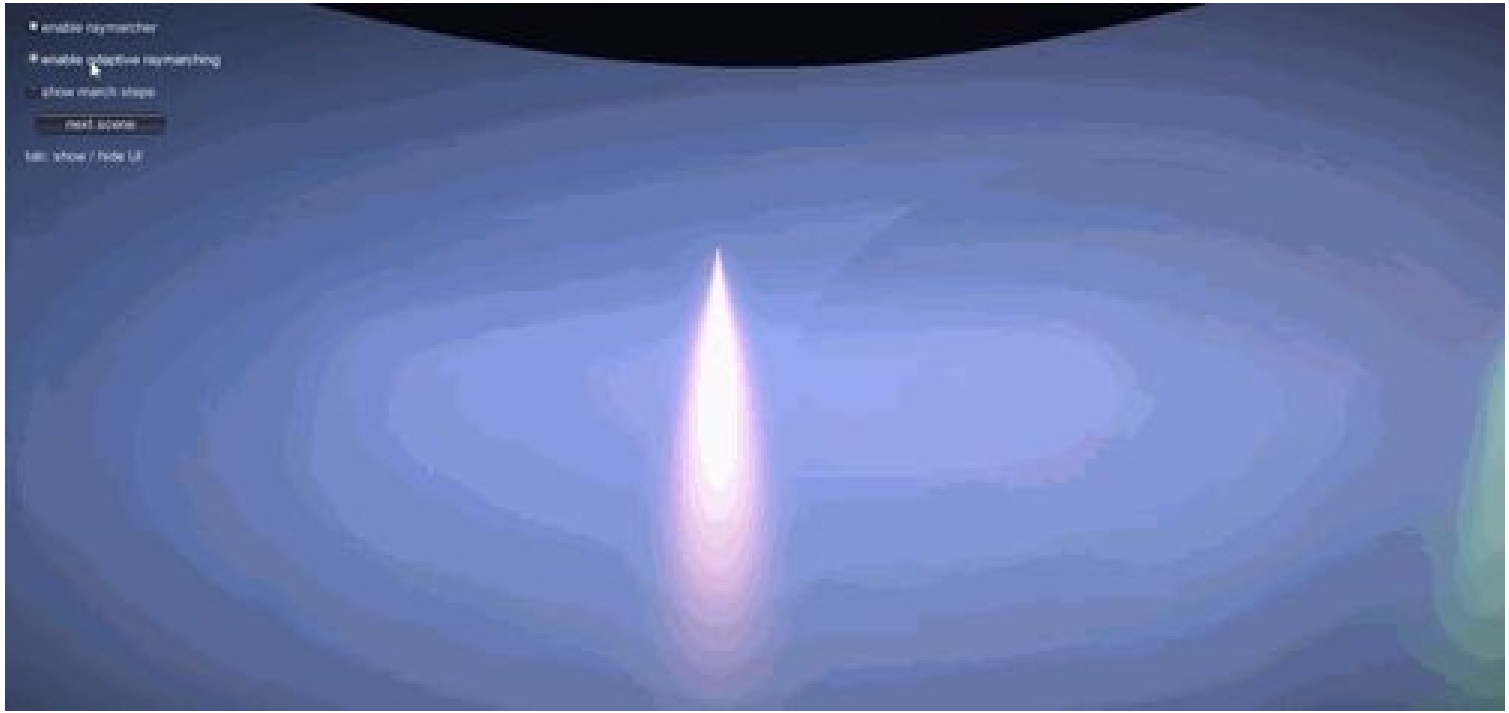
- 非常に上手く機能する
- 品質の低下は少ない方だが、細部が潰れるのが欠点



Temporal Raymarching

- 数フレームに分けて Raymarching することで 1 フレームあたりの Step 数を減らす
  - Step 数少なめで Raymarching
  - 前フレームの Ray の位置を Temporal Reprojection しつつ継続

Temporal Raymarching



Temporal Raymarching

- 大きな高速化が見込めるが、大きな問題も数点
  - 結果が安定するまで輪郭付近が汚くなる
  - Temporal Reprojection の誤差により Ray が貫通する
- 輪郭付近は Step 数多めにする、モーションブラーなどで誤魔化す、Ray を少し引き戻すなど、泥臭い対策が必要

Conservative Depth Output

- Pixel Shader で自力で Depth を出力する場合、Early Depth Test が無効化される
  - 遮蔽される Pixel にも Pixel Shader が走ってしまう

Conservative Depth Output

- Conservative Depth Output でその欠点をある程度克服できる
  - D3D11 世代で備わった機能
  - SV_DepthGreaterEqual / SV_DepthLessEqual
- 出力 Depth に本来よりも 前 / 後ろ の前提を与えられる
- 自力で Depth を出力しつつ Early Depth Test を有効にできる

Conservative Depth Output

- 今回の場合 Depth は Bounding Volume より必ず奥になる
- SV_DepthGreaterEqual で高速化が見込める
- ポリゴンモデルより後に描くようにすればより効果的

その他使えるかもしれないテクニック

- 近隣 Pixel の Ray から Normal 生成
  - ddx() / ddy() で勾配を算出
  - Normal 生成には通常 6 回 Distance Function を計算する必要があるが、そのコストを削減できる
  - そのままでは品質の劣化が激しい

```
1       float3 d1 = ddx(ray_pos);
2       float3 d2 = ddy(ray_pos);
3       float3 normal = normalize(cross(d2, d1));
```

その他使えるかもしれないテクニック

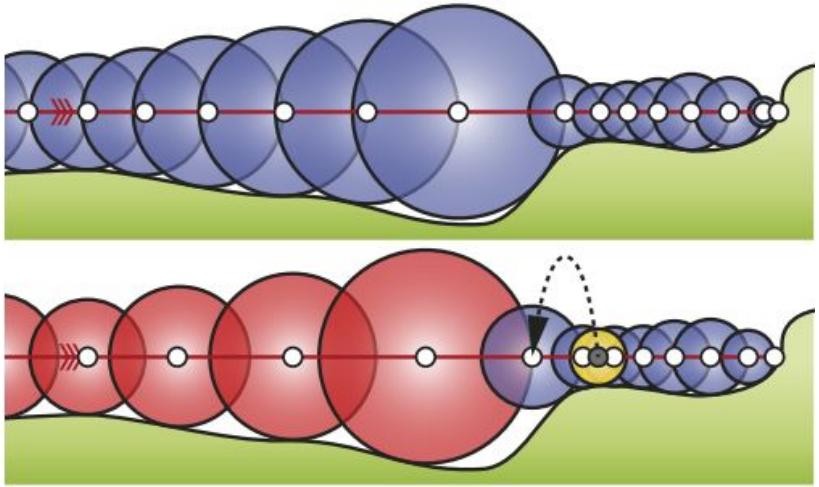
- 横半分の解像度でレンダリングして前フレームの結果と混ぜる
  - Temporal Reprojection で補正しつつ Interlace で合成
  - 効果はそこそこだが品質も落ちる
  - Killzone: Shadow Fall のマルチプレイヤーモードで採られた手法

その他使えるかもしれないテクニック

- Enhanced Sphere Tracing
  - Mercuryによる提案手法
    - http://erleuchtet.org/~cupe/permanent/enhanced_sphere_tracing.pdf
  - intro 用テクニックでゲームにはやや不向き
  - 以下大雑把に紹介

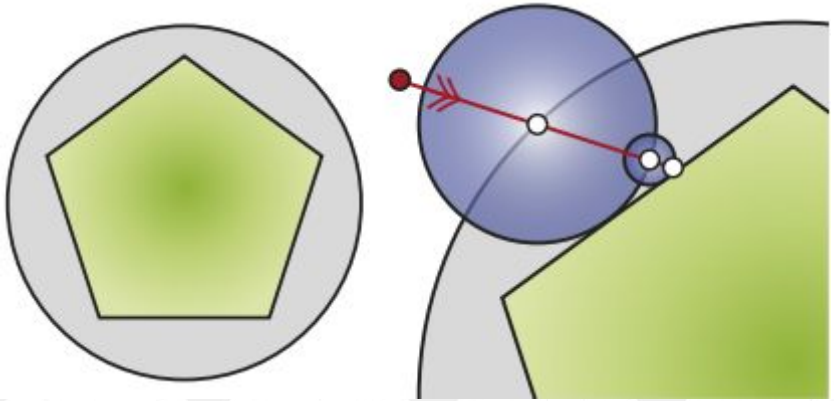
Enhanced Sphere Tracing

- Over-Relaxation Sphere Tracing
  - Ray を通常より多めに進める
  - Sphere がオーバーラップしなかったら引き戻して通常運行



Enhanced Sphere Tracing

- Bounding Volume を用いた最適化
  - Distance Function の Bounding Volume を用意
    - Sphere など Ray との交差点を一発で求められる図形
  - Ray と Bounding Volume との交差点を計算
  - 交差点から Raymarching 開始



その他使えるかもしれないテクニック

- Distance Function を事前にサンプリングして近似モデルを生成

事前サンプリングによる近似モデル

- 複雑で描画面積が多い Distance Function を想定
  - ≒フラクタル
- 事前に Distance Function をサンプリングして近似モデルを用意
  - Marching Cubes 法やボクセルモデル化などで生成
- 近似モデルを Depth のみ専用のバッファに出力
  - 遮蔽されるピクセルに Raymarching が走るのを避けるため
- 出力された Depth を開始点として Raymarching

事前サンプリングによる近似モデル

- 少ない Step 数でいい結果を得られる
- モデルによっては描画 Pixel 数軽減も期待できる
- 効果はあるが、運用が難しい
  - 事前モデル生成が面倒
  - カメラがモデルに突っ込んだら Fullscreen Quad に切り替える必要がある
    - 突っ込んだのをどうやって検出する?
    - CPU 側で Distance Function 計算?

ゲームへの応用: まとめ

- ポリゴンモデルとの混在は問題なく可能
- パーツ分解によりある程度の編集の柔軟性を確保可能
  - エディタ上でパーツを食い合わせてシーンを構築
- 工夫次第でかなり速くできる
  - ただし 速度:品質 のトレードオフは不可避
- 局所的な用途に限れば十分実用レベル

ゲームへの応用: まとめ

- 本公演の作例データなど (Unity 用プロジェクト)
  - https://github.com/i-saint/Unity5Effects

実プロダクションに使われた例

正解するカド (©TOEI ANIMATION)

Questions?

End

ありがとうございました！

Follow us: @TokyoDemoFest

References

Tokyo Demo Fest

http://tokyodemofest.jp/2017/

https://twitter.com/TokyoDemoFest

https://twitter.com/TokyoDemoFestEN

Ctrl-Alt-Test

http://www.ctrl-alt-test.fr/

References

kkrunchy

http://www.farbrausch.de/~fg/kkrunchy/

https://github.com/farbrausch/fr_public/tree/master/kkrunchy

PicoC

https://github.com/zsaleeba/picoc

Tweakable values

https://github.com/joeld42/ld48jovoc/blob/master/ld17_island/tweakval.h

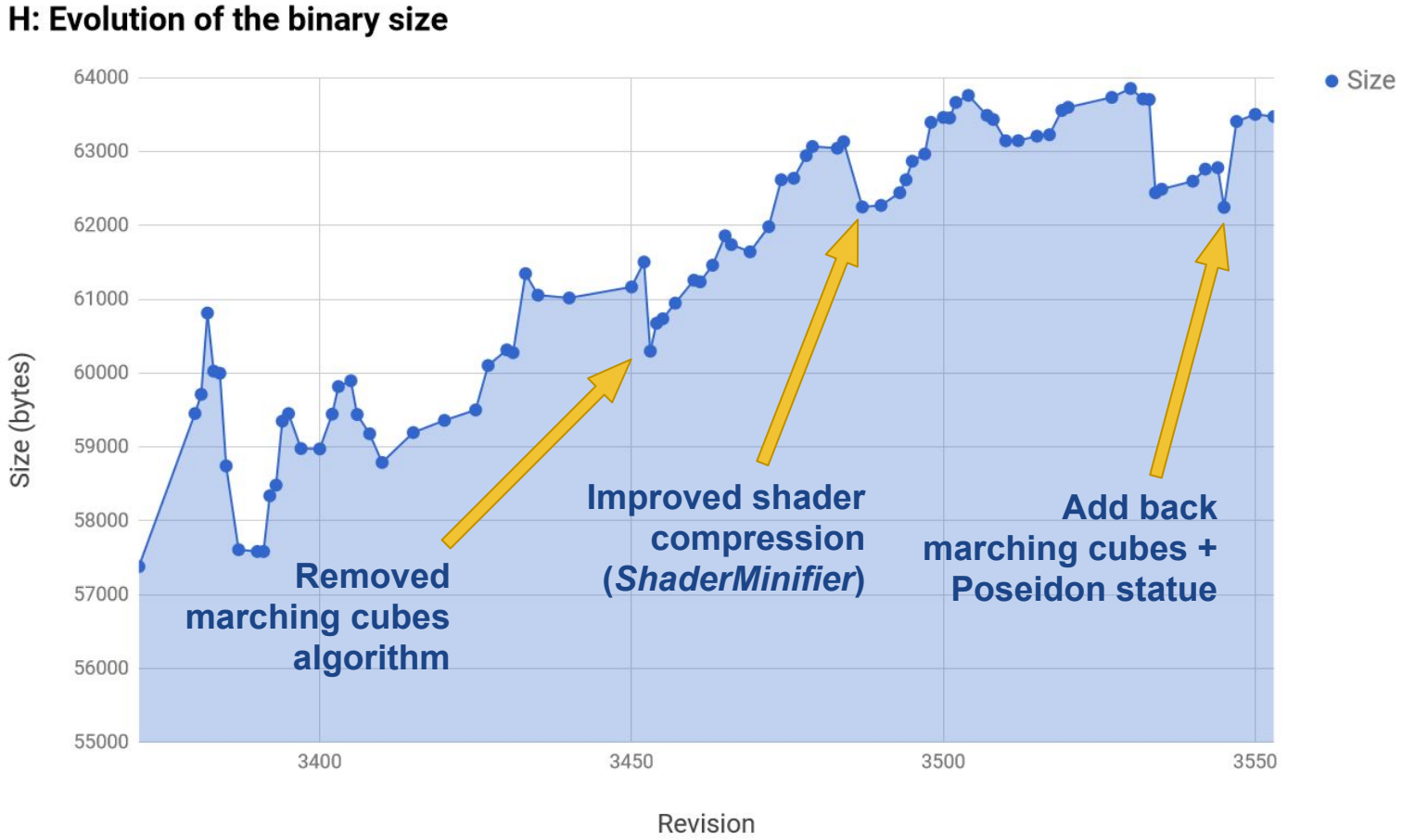
https://www.gamedev.net/resources/_/technical/game-programming/tweakable-constants-r2731

Runtime Compiled C++

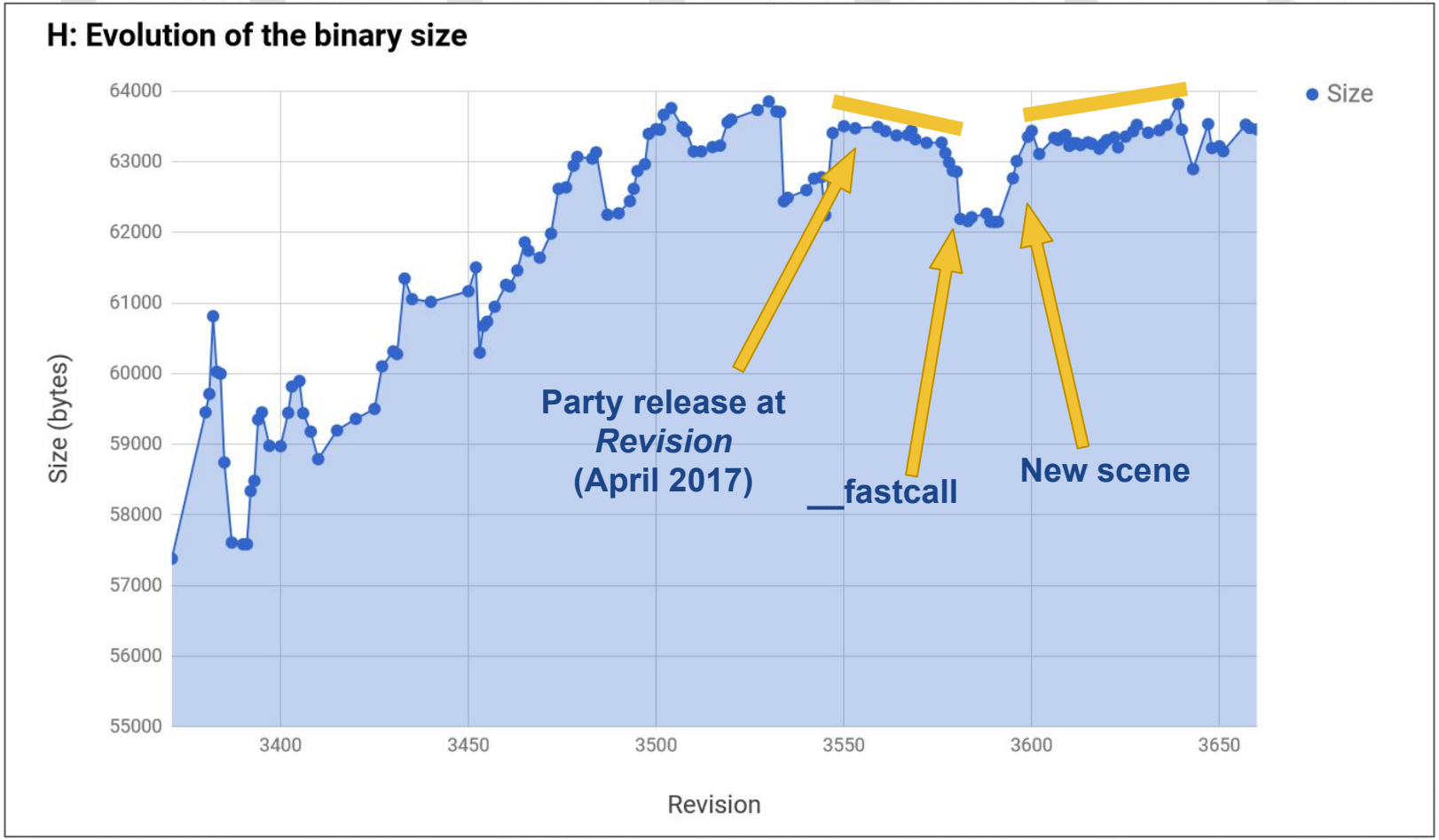
https://github.com/RuntimeCompiledCPlusPlus/RuntimeCompiledCPlusPlus

Bonus slides (Zavie)

サイズについて


H: Evolution of the binary size
Size
Size (bytes)
64000
63000
62000
61000
60000
59000
58000
57000
56000
55000
3400
3450
3500
3550
Revision
Removed marching cubes algorithm
Improved shader compression (*ShaderMinifier*)
Add back marching cubes + Poseidon statue

サイズについて


H: Evolution of the binary size
Size
Size (bytes)
64000
63000
62000
61000
60000
59000
58000
57000
56000
55000
3400
3450
3500
3550
3600
3650
Revision
Party release at
Revision
(April 2017)
__fastcall
New scene

デバッグについて

Syntax highlighting?

- Use meaningful colors.
- Make the invisible visible.
- Make the dangerous visible.

→ Local/global, variable/function, class/namespace, etc.

デバッグについて