

コンピュートシェーダ

2025年度 プログラムワークショップIV (13)

今回のリポジトリ

- https://github.com/tpu-game-2025/PGWS4_13_compute_shader

tpu-game-2025 / PGWS4_13_compute_shader

Code Issues Pull requests Actions Projects Security Insights Settings

PGWS4_13_compute_shader Private

develop had recent pushes 10 seconds ago

main 2 Branches 0 Tags

Go to file t + Code

imagire initialize a5bd035 · 1 minute ago 2 Commits

src initialize 1 minute ago

README.md initialize 1 minute ago

Result.gif initialize 1 minute ago

Result1.png initialize 1 minute ago

Result2.gif initialize 1 minute ago

README

コンピュートシェーダー

はじめに

プログラムワークショップIVの管理用です。

解答はdevelopブランチを見てください。

結果画像

コンピュートシェーダー入門

もくじ

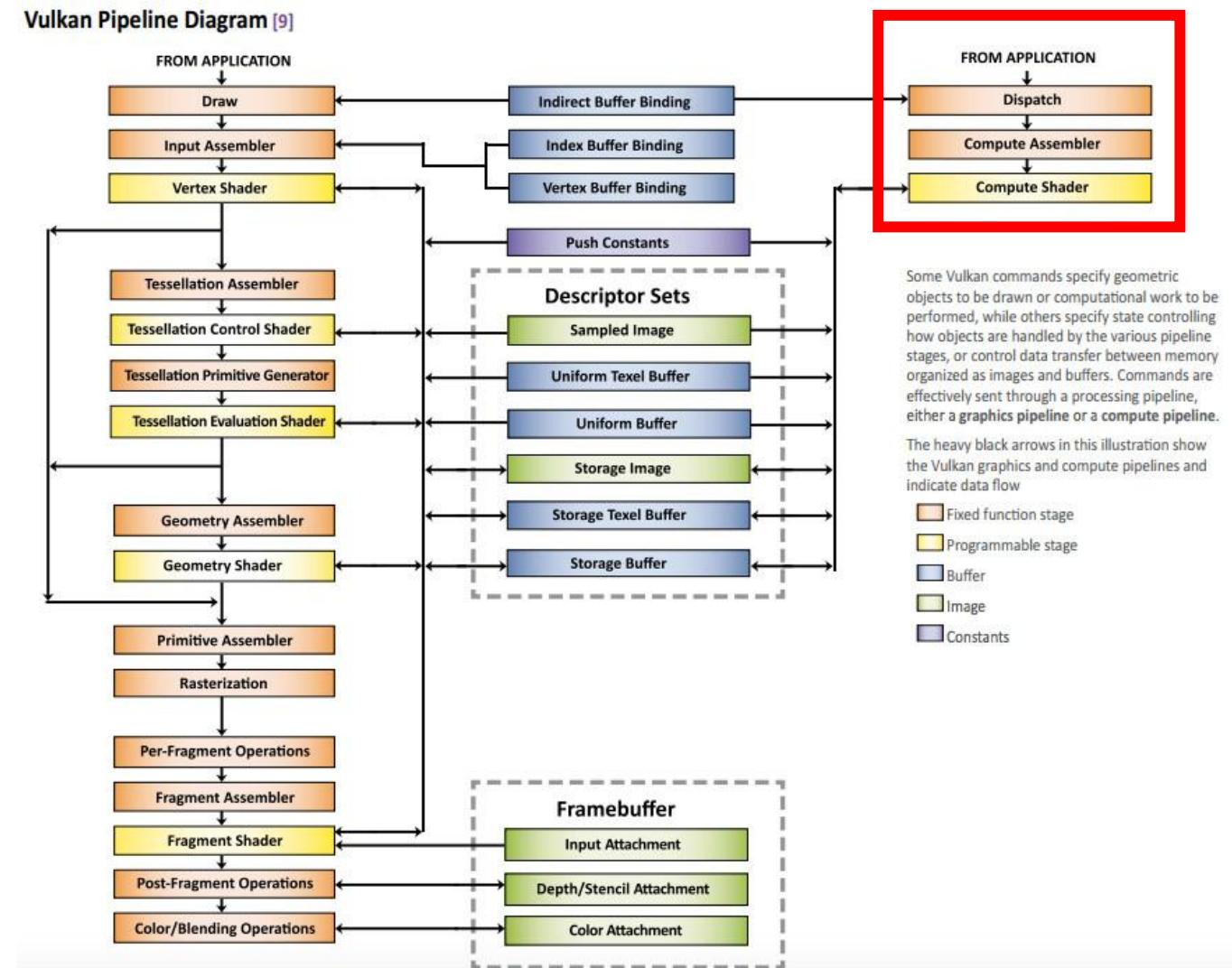
- ・コンピュートシェーダ概要
- ・簡単なコンピュートシェーダ
- ・GPUパーティクル

もくじ

- コンピュートシェーダ概要
- 簡単なコンピュートシェーダ
- GPUパーティクル

コンピュートシェーダ

- 現在、自由度が高い処理に用いられているシェーダ
- 描画目的以外にもGPUの並列性を使えるようにしよう



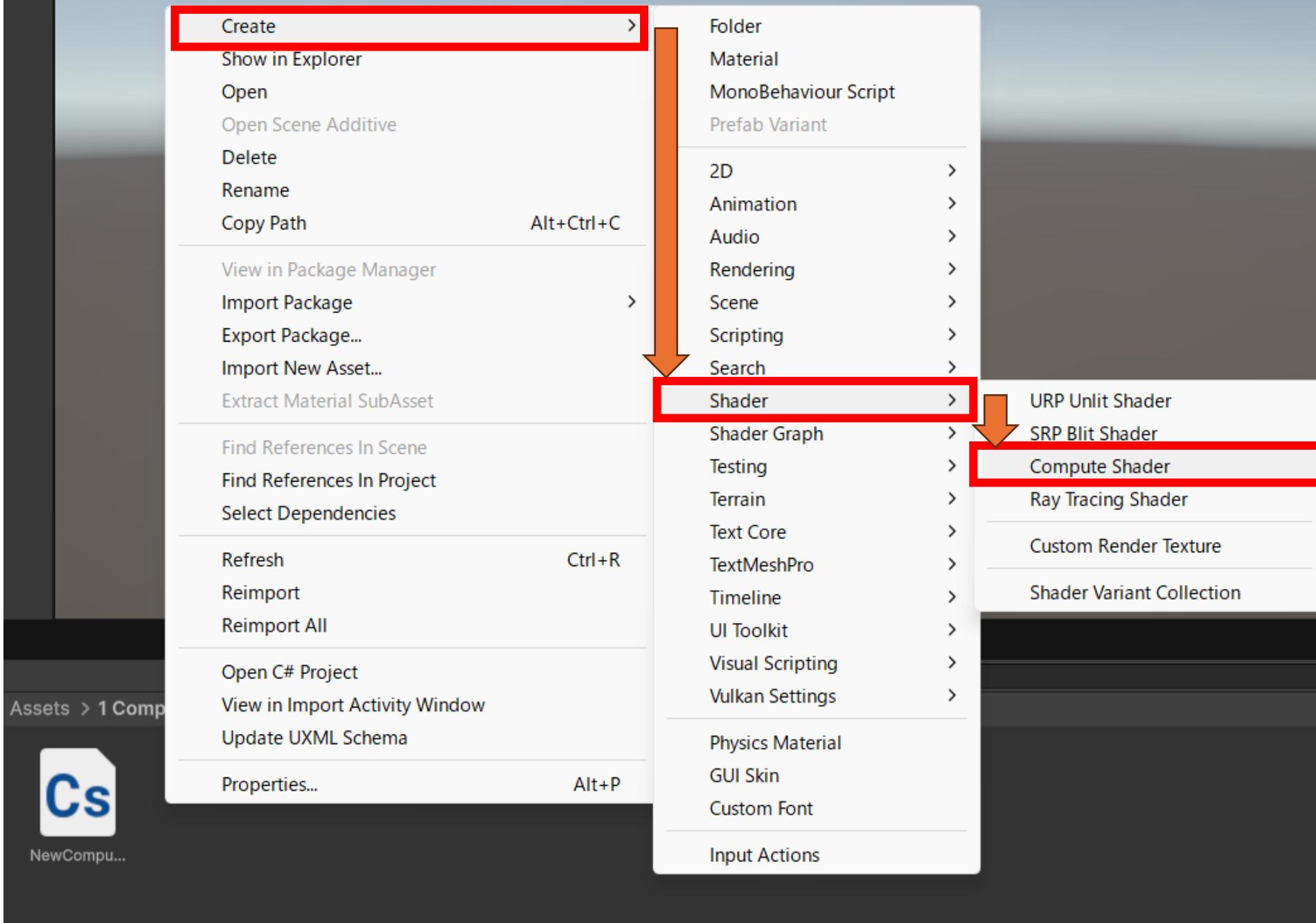
GPUに向いた計算

- 同じプログラムをたくさんのデータに対して適応する
 - 数千から数万の計算コアを有するので、何万以上の個数が向く
 - なるべく独立して、他との依存性がない計算が良い



使い道

- GPUパーティクル
 - 細かなパーティクル
 - 破片の物理演算
- Cluster Shading
 - 空間をブロックに分け、ブロックごとに描画するオブジェクト・ライトを洗い出して、大量のライトのシーンを高速に描画する



自動で追加されるソースコードの中身

```
// それぞれの#kernelは、どの関数をコンパイルするかを指示します;多くのカーネルを持てます
1  // Each #kernel tells which function to compile; you can have many kernels
2  #pragma kernel CSMain
3  // enableRandomWriteフラグを持つRenderTextureを作成し、cs.SetTextureで設定せよ
4  // Create a RenderTexture with enableRandomWrite flag and set it
5  // with cs. SetTexture
6  RWTexture2D<float4> Result; 4成分floatの読み書きできるテクスチャ
7
8  [numthreads(8, 8, 1)] 8x8x1の単位で実行数(64単位で実行される)
9  // 戻り値なし void CSMain (uint3 id : SV_DispatchThreadID) 全スレッドを通してユニークなIDを受ける
10 { // やること: 実際のコードをここに挿入せよ!
11   // TODO: insert actual code here!  てきとうな計算
12
13   Result[id.xy] = float4(id.x & id.y, (id.x & 15) / 15.0, (id.y & 15) / 15.0, 0.0);
14 }
```

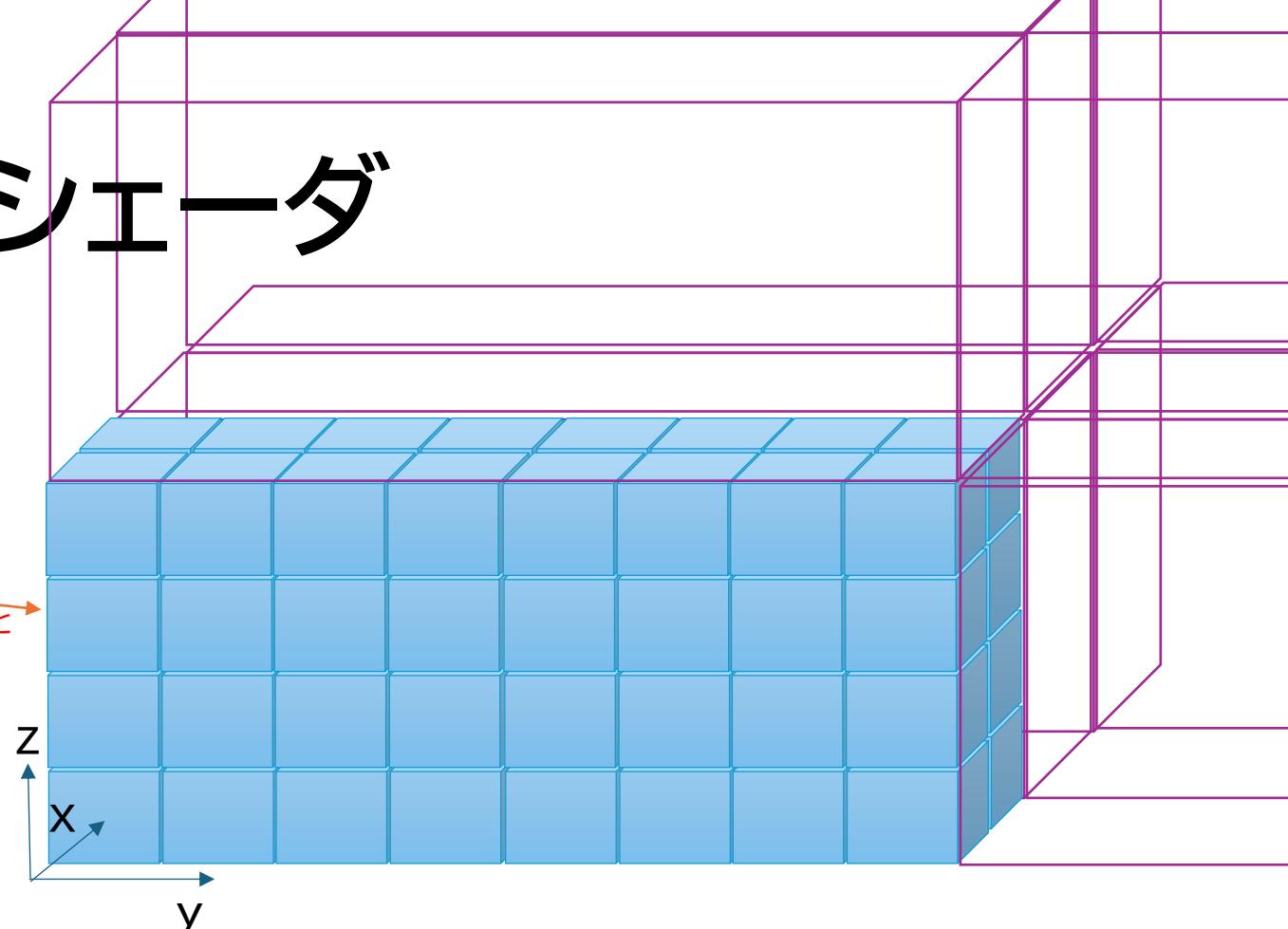
ビット計算例 X軸で16個の周期計算 Y軸で16個の周期計算

簡単なコンピュートシェーダ

```
#pragma kernel 関数名
```

```
RWTexture3D<float4> tex;
```

```
[numthreads(2, 8, 4)] これらの積が32や64の倍数でないと  
GPUで暇になるコアが発生する
void 関数名(uint3 id : SV_DispatchThreadID)
{
    tex[id] = float4(0, 0, 0, 1);
}
```

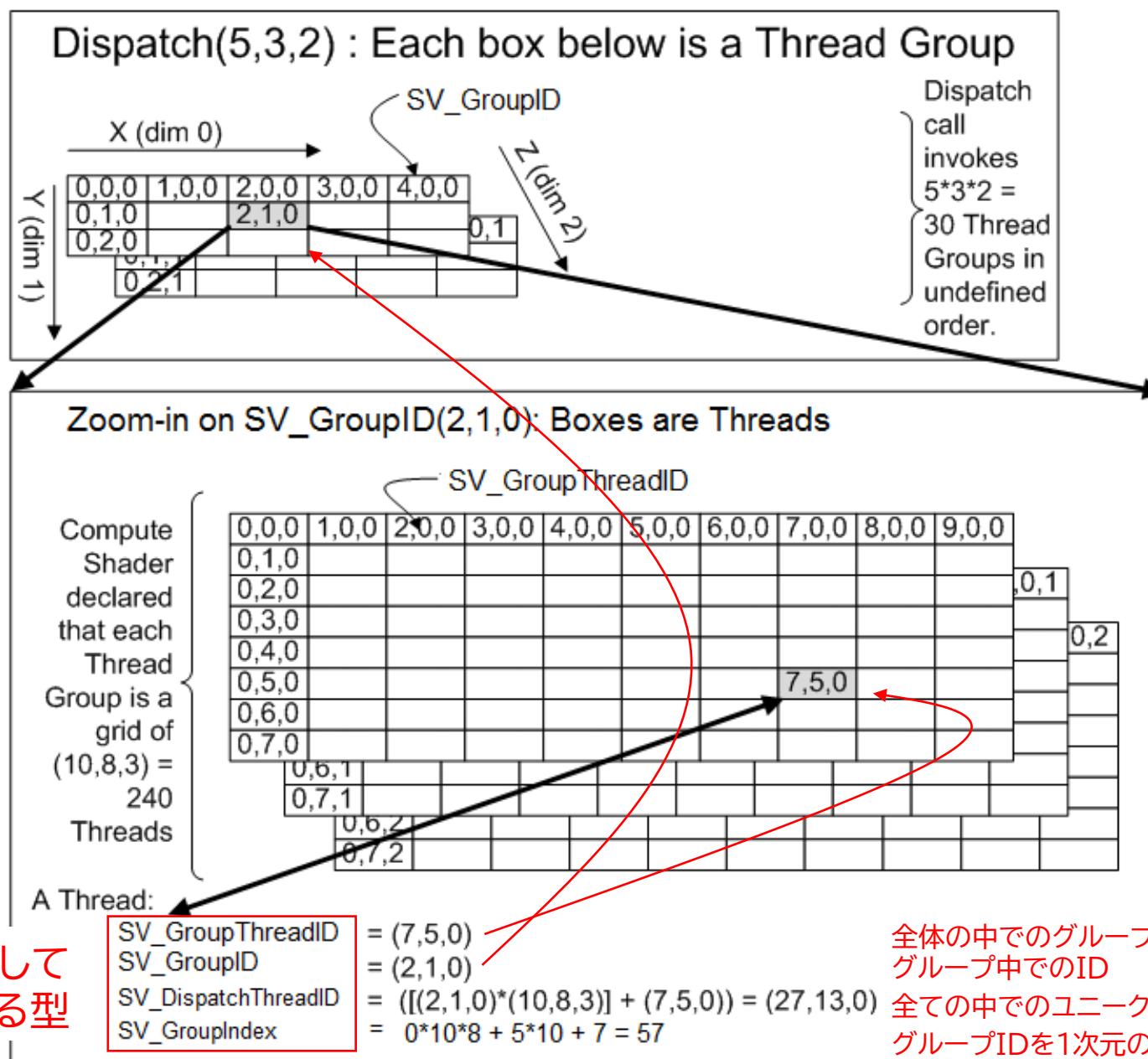


C#側

```
int kernel = computeshader.FindKernel("関数名");
computeshader.Dispatch(kernel, 3, 5, 7); ➔ (3*2, 5*8, 7*4) = (6, 40, 28) = 6720個の処理が走る
```

3次元配列の入れ子構造

<https://docs.microsoft.com/ja-jp/windows/win32/api/d3d11/nf-d3d11-id3d11devicecontext-dispatch>



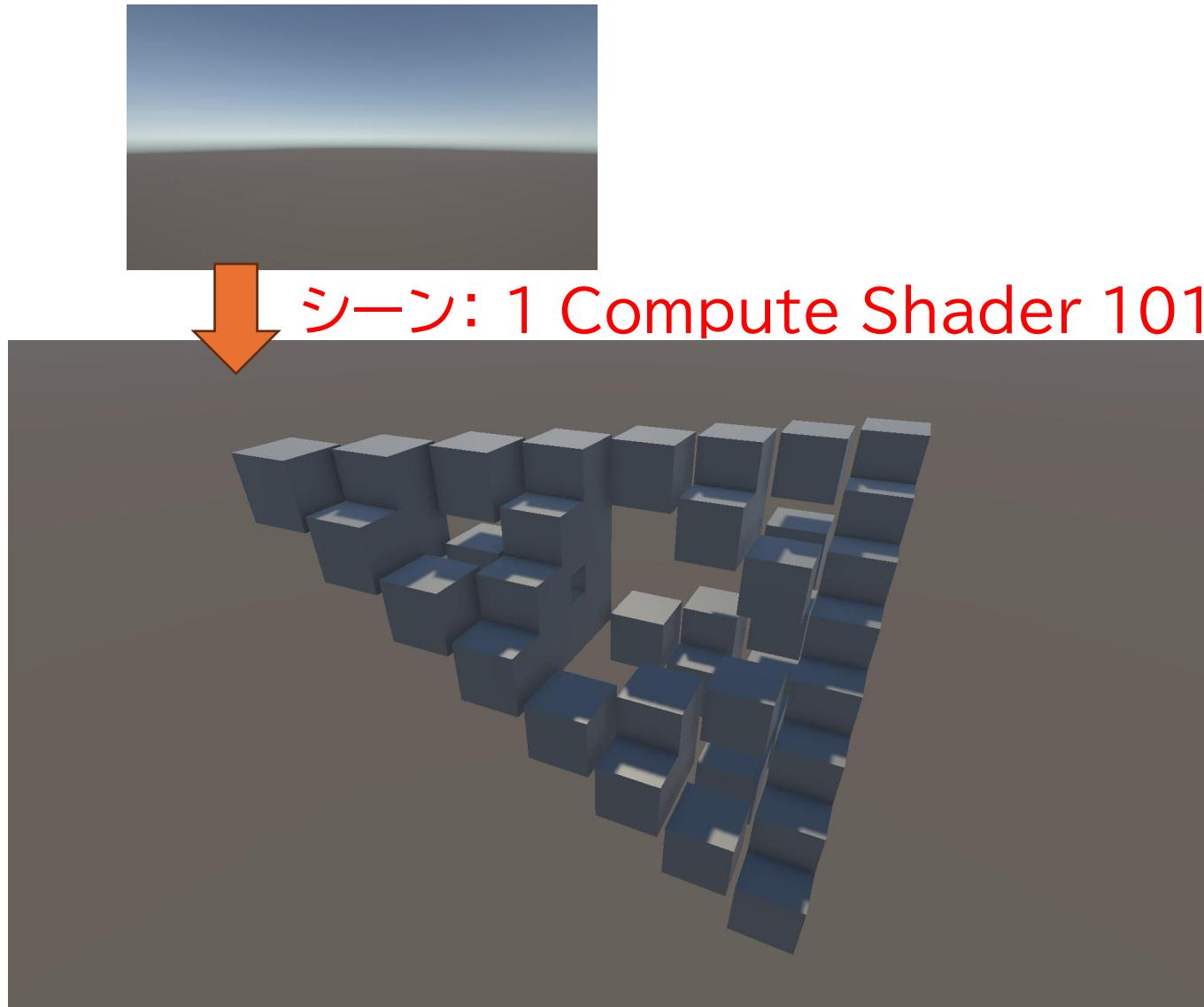
引数として
取りうる型

グローバル変数に使えるデータ型

- 読み書き可能(非圧縮、遅い)
 - RWByteAddressBuffer:配列
 - RWStructuredBuffer:構造体配列
 - RWTexture1D
 - RWTexture1DArray:配列
 - RWTexture2D
 - RWTexture2DArray
 - RWTexture3D
- 読み込み専用(圧縮されているかも?速い)
 - StructuredBuffer
 - Texture1D
 - Texture1DArray
 - Texture2D
 - Texture2DArray
 - Texture3D
 - Texture2DMS:マルチサンプリング
 - Texture2DMSArray
 - TextureCube:キューブマップ
 - TextureCubeArray

本日の内容

- ・コンピュートシェーダ概要
- ・簡単なコンピュートシェーダ
 - ・GPUの計算結果をCPUで読み込んで、オブジェクトを配置してみる
- ・GPUパーティクル



最低限のコード変更

- RWTexture2DをCPUで取得するのが面倒なので型を変更

```
1 // Each #kernel tells which function to compile; you can have many kernels
2 #pragma kernel CSMain
3
4 // Create a RenderTexture with enableRandomWrite flag and set it
5 // with cs. SetTexture
6 // ↓ RWTexture2D<float4> Result;
7 RWStructuredBuffer<float4> Result;
8
9 [numthreads(8, 8, 1)]
10 void CSMain (uint3 id : SV_DispatchThreadID)
11 {
12     // TODO: insert actual code here!
13
14     // ↓ Result[id.xy] = float4(id.x & id.y, (id.x & 15) / 15.0, (id.y & 15) / 15.0, 0.0);
15     Result[id.y*8+id.x] = float4(id.x & id.y, (id.x & 15) / 15.0, (id.y & 15) / 15.0, 0.0);
16 } インデックスを1次元に変更
```

オブジェクト配置

ComputeShader MonoBehaviour Scriptを追加して編集

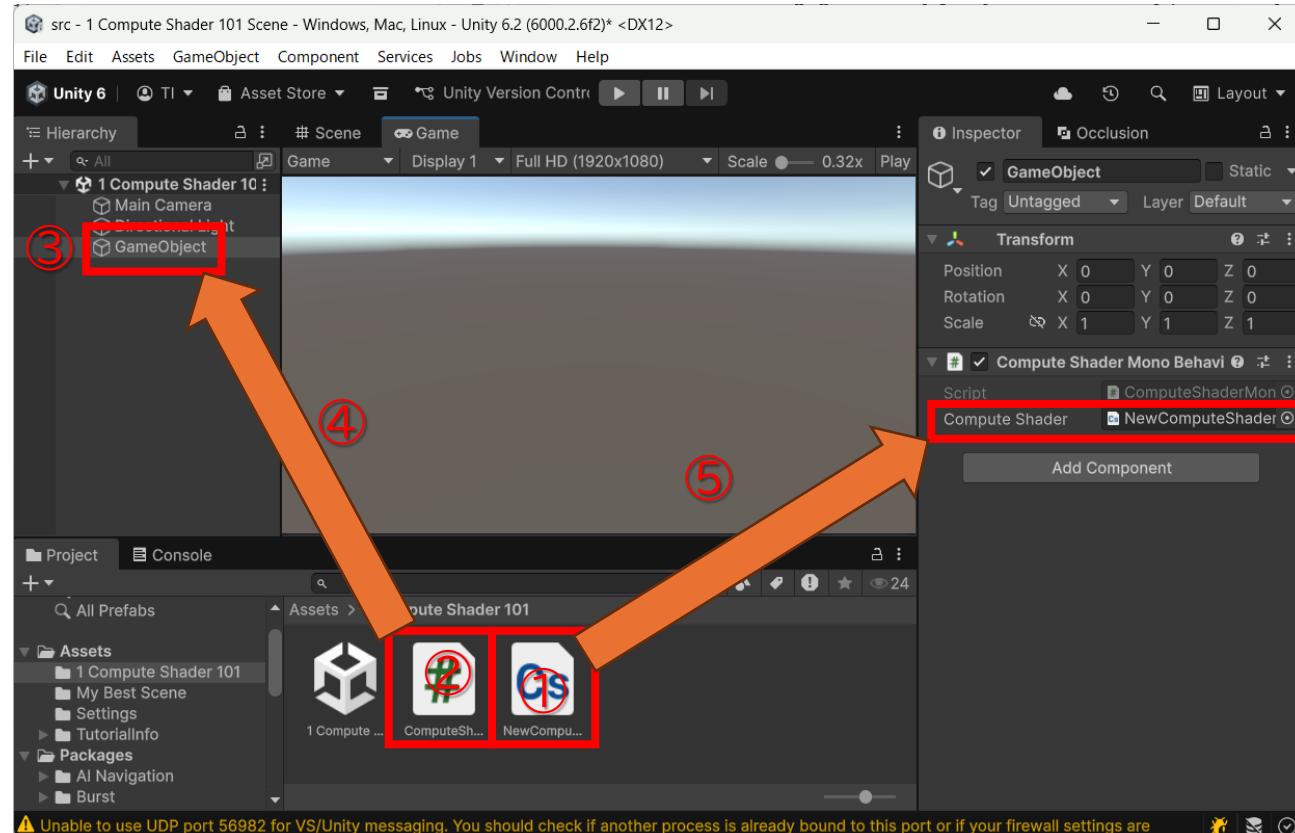
- ・ 今回は、8x8個実行
- ・ ComputeBuffer
 - ・ RWStructuredBuffer(シェーダ側)のコンテナ
- ・ Dispatch:実行
- ・ GetData
 - ・ CPUが読めるメモリにコピー
- ・ CreatePrimitive
 - ・ インスタンス生成

```
1  using UnityEngine;
2
3  ① Unity スクリプト (1 件のアセット参照) | 0 個の参照
4  public class ComputeShaderMonoBehaviourScript : MonoBehaviour
5  {
6
7      ② [SerializeField] ComputeShader computeShader = default!;
8
9      ③ // Start is called once before the first execution of Update after the MonoBehaviour is created.
10     ④ [Unity メッセージ | 0 個の参照]
11     void Start()
12     {
13
14         ⑤ int x = 8;
15         ⑥ int y = 8;
16         ⑦ ComputeBuffer computeBuffer = new ComputeBuffer(4 * x * y, sizeof(float));
17
18         ⑧ int kernelHandle = computeShader.FindKernel("CSMain"); // カーネル関数のハンドルを取得
19         ⑨ computeShader.SetBuffer(kernelHandle, "Result", computeBuffer); // バッファをセット
20         ⑩ computeShader.Dispatch(kernelHandle, x / 8, y / 8, 1); // カーネル関数を実行
21
22         ⑪ float[] result = new float[4 * x * y];
23         ⑫ computeBuffer.GetData(result); // CPUからGPUへデータを転送
24         ⑬ computeBuffer.Release(); // バッファの解放
25
26         ⑭ // 確認用に立方体を表示してみる
27         ⑮ for (int i = 0; i < x; i++)
28         {
29             ⑯ for (int j = 0; j < y; j++)
30             {
31
32                 ⑰ float cs_x = result[4 * (i + j * x) + 0];
33                 ⑱ float cs_y = result[4 * (i + j * x) + 1] * 10.0f;
34                 ⑲ float cs_z = result[4 * (i + j * x) + 2] * 10.0f;
35                 ⑳ GameObject cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
36                 ㉑ cube.transform.position = new Vector3(cs_x, cs_y, cs_z);
37                 ㉒ cube.transform.localScale = new Vector3(0.9f, 0.9f, 0.9f); // 隙間をあける
38             }
39         }
40     }
41 }
```

やってみよう：

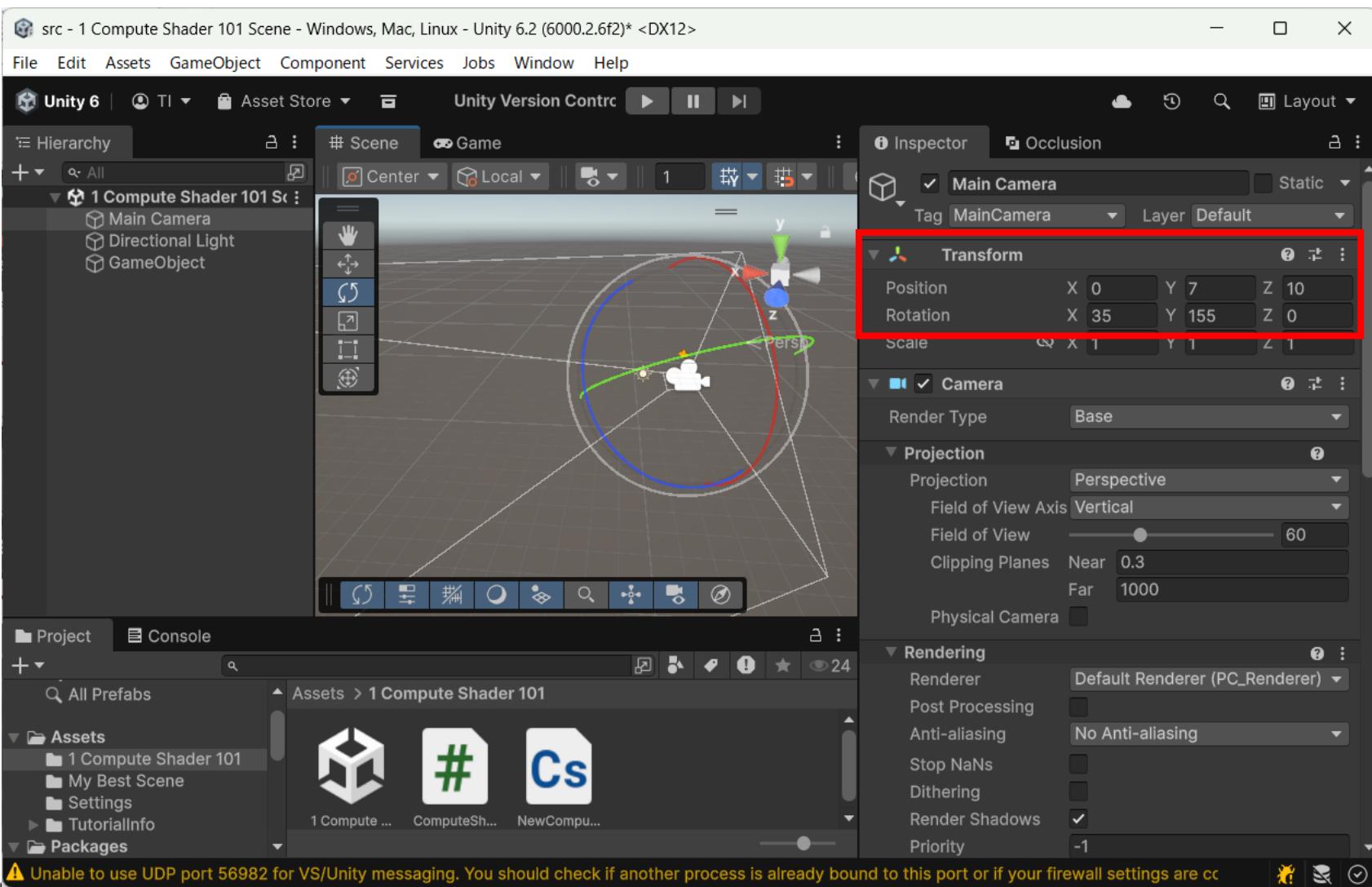
1 Compute Shader 101/1 Compute Shader 101 Scene

1. Compute Shader を追加
 - ・名称例: NewComputeShader
2. C# Scriptを追加
 - ・名称例: ComputeShader MonoBehaviourScript
 - ・C# Scriptを書き換え
 - ・Compute Shader を書き換え
3. EmptyなGameObjectを追加
 - ・名称例: GameObject
4. GameObject にCompute ShaderMonoBehaviour Scriptをバインド
5. ComputeShaderMonoBehaviourScriptに NewComputeShaderを設定

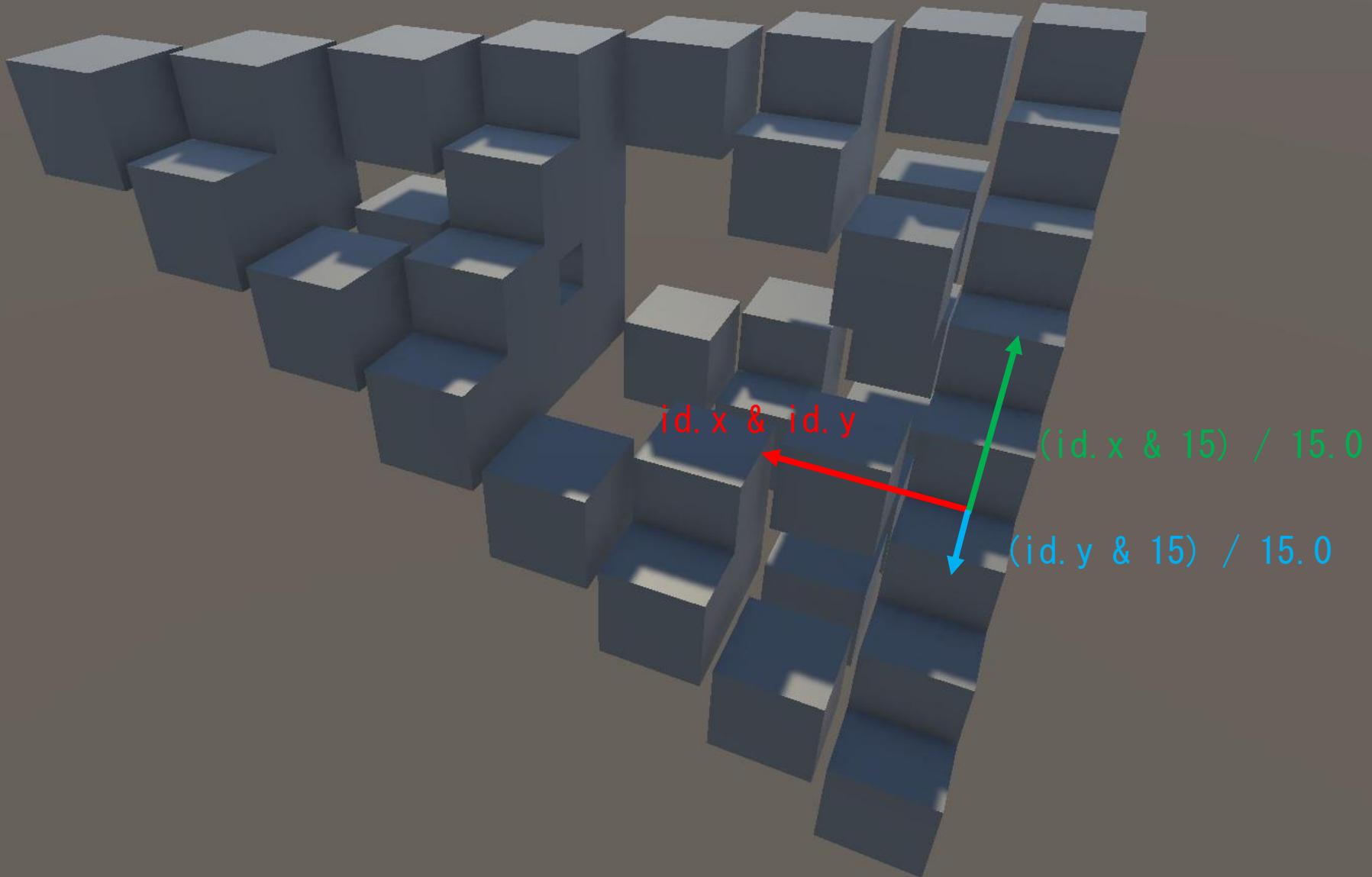


カメラを移動

- 何となく見やすい位置に

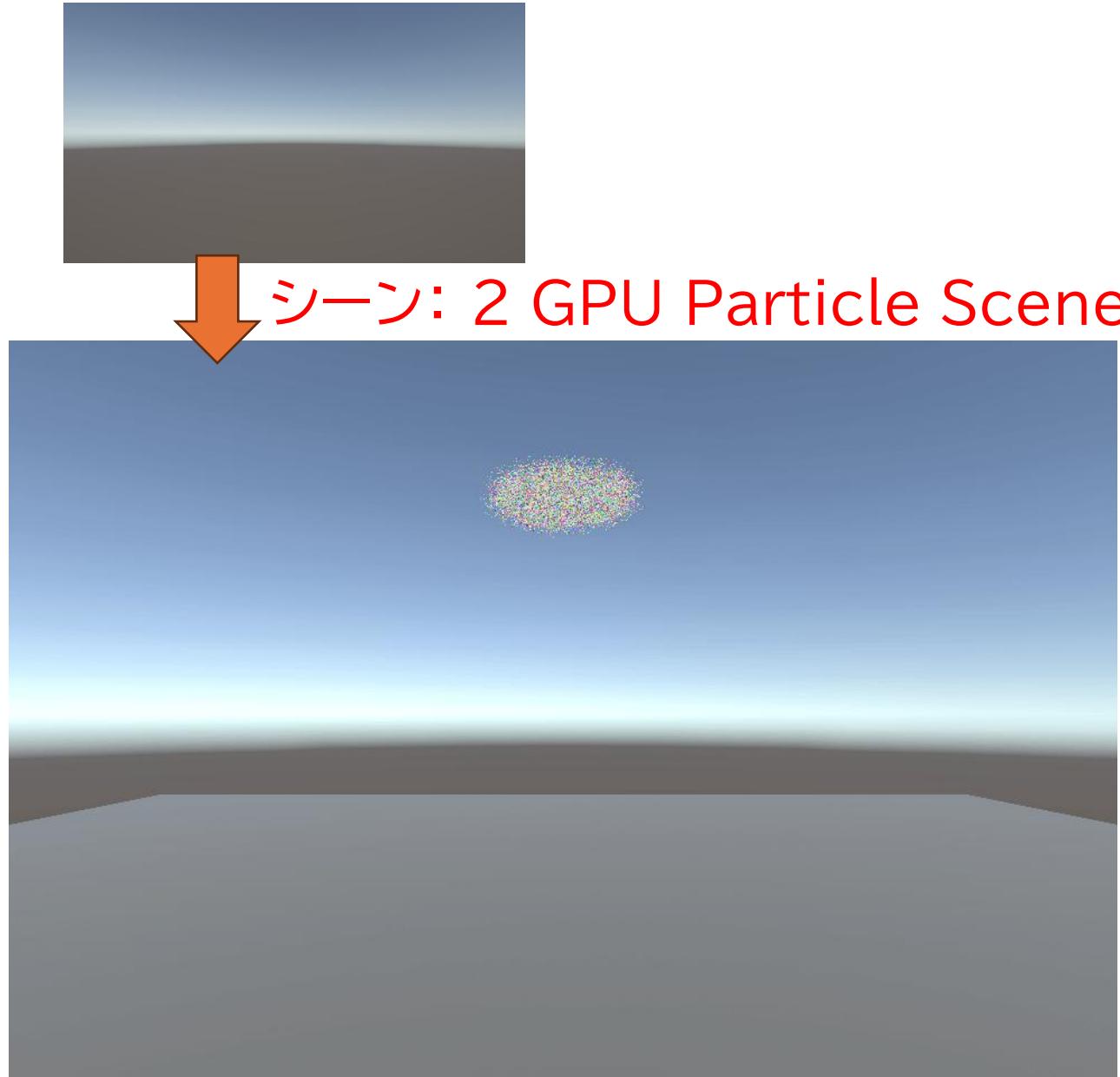


完成



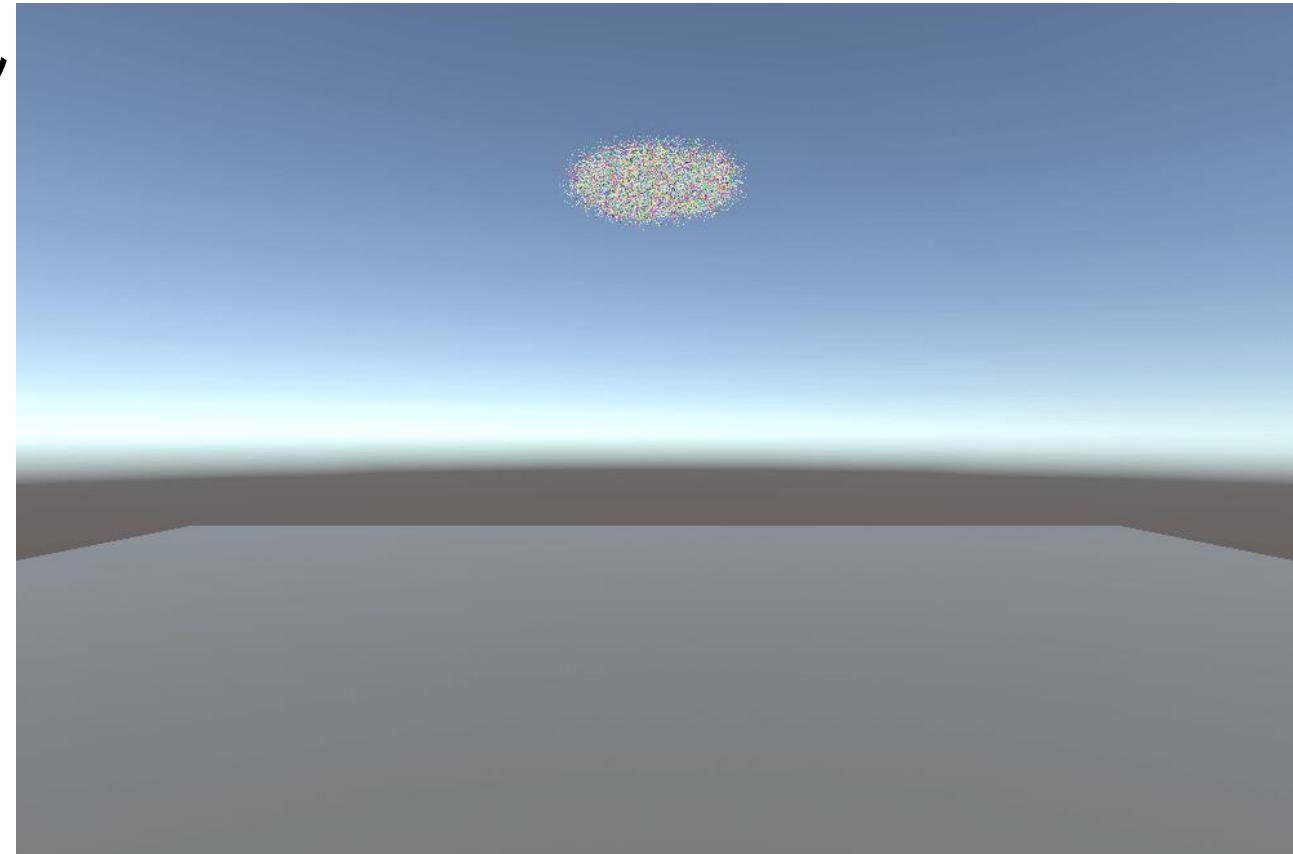
もくじ

- コンピュートシェーダ概要
- 簡単なコンピュートシェーダ
- GPUパーティクル



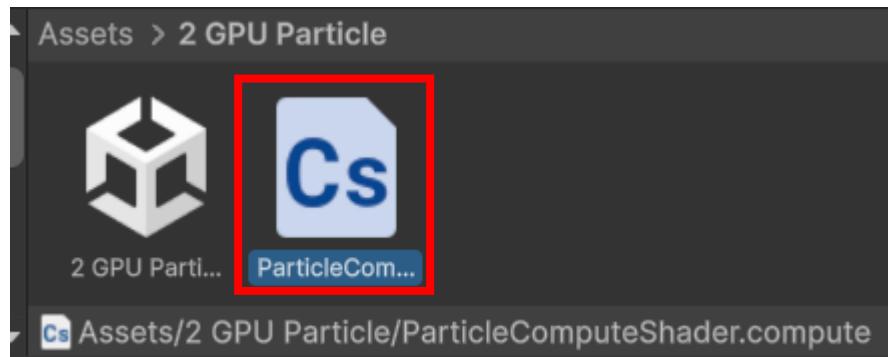
GPUパーティクルを試す

- 地面についたら反射
- 動きが止まつたら吹き出し直し
- 噴水のように動く
- 色はランダム



コンピュートシェーダの追加

- ・名称例: ParticleComputeShader



乱数生成

ParticleComputeShader.compute

```
15 // 亂数関数(see. https://andantesoft.hatenablog.com/entry/2024/12/19/193517)
16 float fihash_orig(float2 v)
17 {
18     uint2 u = asint(v * float2(141421356, 2718281828));
19     return float((u.x ^ u.y) * 3141592653) * 2.3283064365386962890625e-10;
20 }
21
22 float2 rand2(float2 st)
23 {
24     return float2(fihash_orig(st), fihash_orig(st + 1));
25 }
26
27 float3 rand3(float2 st)
28 {
29     return float3(fihash_orig(st), fihash_orig(st + 1), fihash_orig(st + 2));
30 }
```



構造体とオブジェクト

- 構造体配列を定義

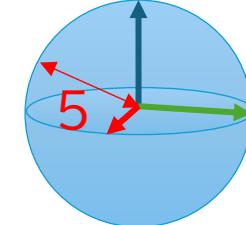
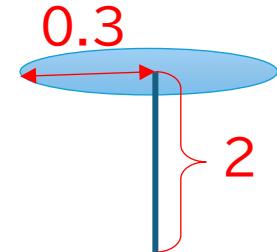
ParticleComputeShader.compute

```
1 // Each #kernel tells which function to compile; you can have many kernels
2 #pragma kernel CSInitialize
3 #pragma kernel CSUpdate 関数の宣言
4
5 struct Particle
6 {
7     float3 position;
8     float3 velocity; 一つ一つの粒子のデータ
9     float3 color;
10 };
11
12 RWStructuredBuffer<Particle> Particles; 構造体配列
13 float deltaTime; // 更新時にCPUから受け取る前フレームからの経過時間
```

初期化

```
32     void reset(uint id)
33     {
34         float2 seed = float2(
35             (1.0 / 256.0) * (float) (id % 256),
36             (1.0 / 256.0) * (float) ((id / 256) % 256)); IDごとに異なる2次元の値を生成
37
38         // 高さ2の半径0.3の円盤上に生成
39         float2 r2 = rand2(seed);
40         Particles[id].position = float3(
41             cos(r2.x * 2.0 * 3.14159265) * r2.y * 0.3,
42             2.0,
43             sin(r2.x * 2.0 * 3.14159265) * r2.y * 0.3
44         );
45
46         // 速度は、球的に広げる
47         float3 r3 = rand3(seed + 10);
48         Particles[id].velocity = float3(
49             5 * (r3.x + 0.01) * sin(r3.y * 3.14159265) * cos(r3.z * 2.0 * 3.14159265),
50             5 * (r3.x + 0.01) * sin(r3.y * 3.14159265) * sin(r3.z * 2.0 * 3.14159265),
51             5 * (r3.x + 0.01) * cos(r3.y * 3.14159265)
52         );
53
54         // 色はランダム
55         Particles[id].color = rand3(seed + 20);
56     }
57
58     #define THREAD_NUM 64
59
60     [numthreads(THREAD_NUM, 1, 1)]
61     void CSInitialize(uint3 id : SV_DispatchThreadID)
62     {
63         reset(id.x);
64     }
```

ParticleComputeShader.compute



運動

- 速度から位置を更新
- 加速度(重力)で速度を更新
 - 空気抵抗で少しずつ遅くする
- 当たり判定
 - 床より下に来たら上向きにする
 - 跳ね返り係数を付けて勢いを落とす
- 終了判定
 - 速度が一定以下になつたら再初期化

```

66 [numthreads(THREAD_NUM, 1, 1)]
67 void CSUpdate(uint3 id : SV_DispatchThreadID)
68 {
69     float3 position = Particles[id.x].position;
70     float3 velocity = Particles[id.x].velocity;
71
72     // 動かなくなったらリセット
73     if (dot(velocity, velocity) < 0.001)
74     {
75         reset(id.x);
76         return;
77     }
78
79     // 下に落ちたら跳ね返る
80     if (position.y < 0.0)
81     {
82         velocity.y = -0.6 * velocity.y;
83     }
84
85     // 陽オイラー
86     position += velocity * deltaTime;
87     velocity.y -= 9.8 * deltaTime;
88     velocity *= 0.995; // 簡易空気抵抗
89
90     // 値の格納
91     Particles[id.x].position = position;
92     Particles[id.x].velocity = velocity;
93 }

```

描画用のアセットを追加

1. マテリアル

- ・名称例: 2 Particle Material

2. シェーダ

- ・「Shader」-「URP Unlit Shader」
- ・2 Particle Materialに設定
- ・名称例: ParticleUnlitUniversalRenderPipelineShader



シェーダ

- URP向けのシェーダとして、ある程度自動的に記述される

ParticleUnlitUniversalRenderPipelineShader. shader

```
1   Shader "Custom/ParticleUnlitUniversalRenderPipelineShader"
2   {
3     SubShader
4     {
5       Tags { "RenderType" = "Opaque" "RenderPipeline" = "UniversalPipeline" }
6
7       Pass
8       {
9         HLSLPROGRAM
10
11         #pragma vertex vert
12         #pragma fragment frag
13
14         #include "Packages/com.unity.render-pipelines.universal/ShaderLibrary/Core.hlsl"
```

パーティクルの描画

- Id値から構造体配列を読み込み位置や色として使用する

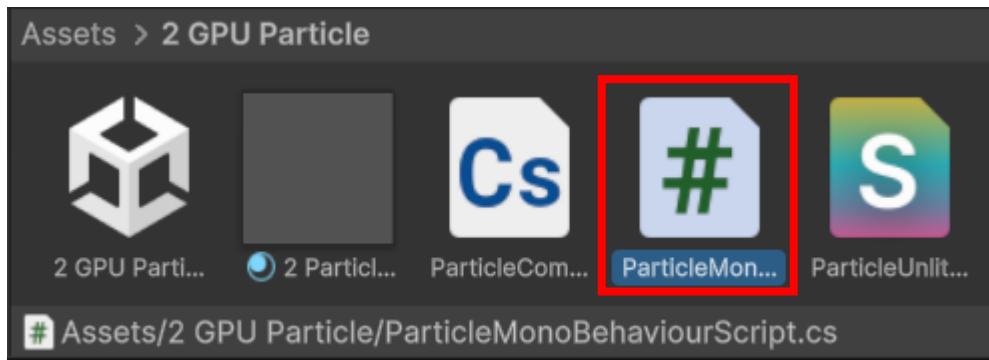
```
16 struct Particle ParticleUnlitUniversalRenderPipelineShader.shader
17 {
18     float3 Position;
19     float3 Velocity;
20     float3 Color;
21 };
22
23 struct Varyings
24 {
25     float4 positionHCS : SV_POSITION;
26     half3 color : COLOR0;
27 };
28
29 uniform StructuredBuffer<Particle> Particles;
30
31 Varyings vert(uint id : SV_VertexID) IDを入力として受け取ることにする
32 {
33     Varyings OUT;
34
35     Particle particle = Particles[id];
36     OUT.positionHCS = TransformObjectToHClip(particle.Position);
37
38     OUT.color = particle.Color;
39     return OUT;
40 }
41
42 half4 frag(Varyings IN) : SV_Target
43 {
44     return half4(IN.color, 1);
45 }
46 ENDHLSL
```

Compute shaderと同じ構造体を定義する

構造体配列へのアクセス

CPU側の処理

- MonoBehaviourスクリプトの追加
 - 名称例: ParticleMonoBehaviourScript



```

1  using UnityEngine;
2  using System.Runtime.InteropServices;
3
4  1 個の参照
5  struct Particle
6  {
7      public Vector3 Position;
8      public Vector3 Velocity;
9      public Vector3 Color;
10 }
11
12  ④ Unity スクリプト (1 件のアセット参照) | 0 個の参照
13  public class ParticleMonoBehaviourScript : MonoBehaviour
14  {
15      [SerializeField] Material material = default!;          表示するシェーダへのアクセス
16      [SerializeField] ComputeShader computeShader = default!;  コンピュートシェーダへのアクセス
17
18      private int updateKernel;  コンピュートシェーダの関数を呼び出すための対応付け
19      private ComputeBuffer buffer;  構造体配列の実体を保持
20
21      private const int THREAD_NUM = 64; // 1つのグループあたりのスレッド数
22      private const int PARTICLE_NUM = ((65536 + THREAD_NUM - 1) / THREAD_NUM) * THREAD_NUM;
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
727
728
729
729
730
731
732
733
734
735
736
737
737
738
739
739
740
741
742
743
744
745
746
746
747
748
748
749
749
750
751
752
753
754
755
756
757
757
758
759
759
760
761
762
763
764
765
766
766
767
768
768
769
769
770
771
772
773
774
775
776
777
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
827
828
829
829
830
831
832
833
834
835
836
837
837
838
839
839
840
841
842
843
844
845
846
846
847
848
848
849
849
850
851
852
853
854
855
856
857
857
858
859
859
860
861
862
863
864
865
866
866
867
868
868
869
869
870
871
872
873
874
875
876
877
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
907
908
909
909
910
911
912
913
914
915
916
916
917
918
918
919
919
920
921
922
923
924
925
926
927
927
928
929
929
930
931
932
933
934
935
936
937
937
938
939
939
940
941
942
943
944
945
946
946
947
948
948
949
949
950
951
952
953
954
955
956
957
957
958
959
959
960
961
962
963
964
965
966
966
967
968
968
969
969
970
971
972
973
974
975
976
977
977
978
979
979
980
981
982
983
984
985
986
986
987
988
988
989
989
990
991
992
993
994
995
996
997
998
999

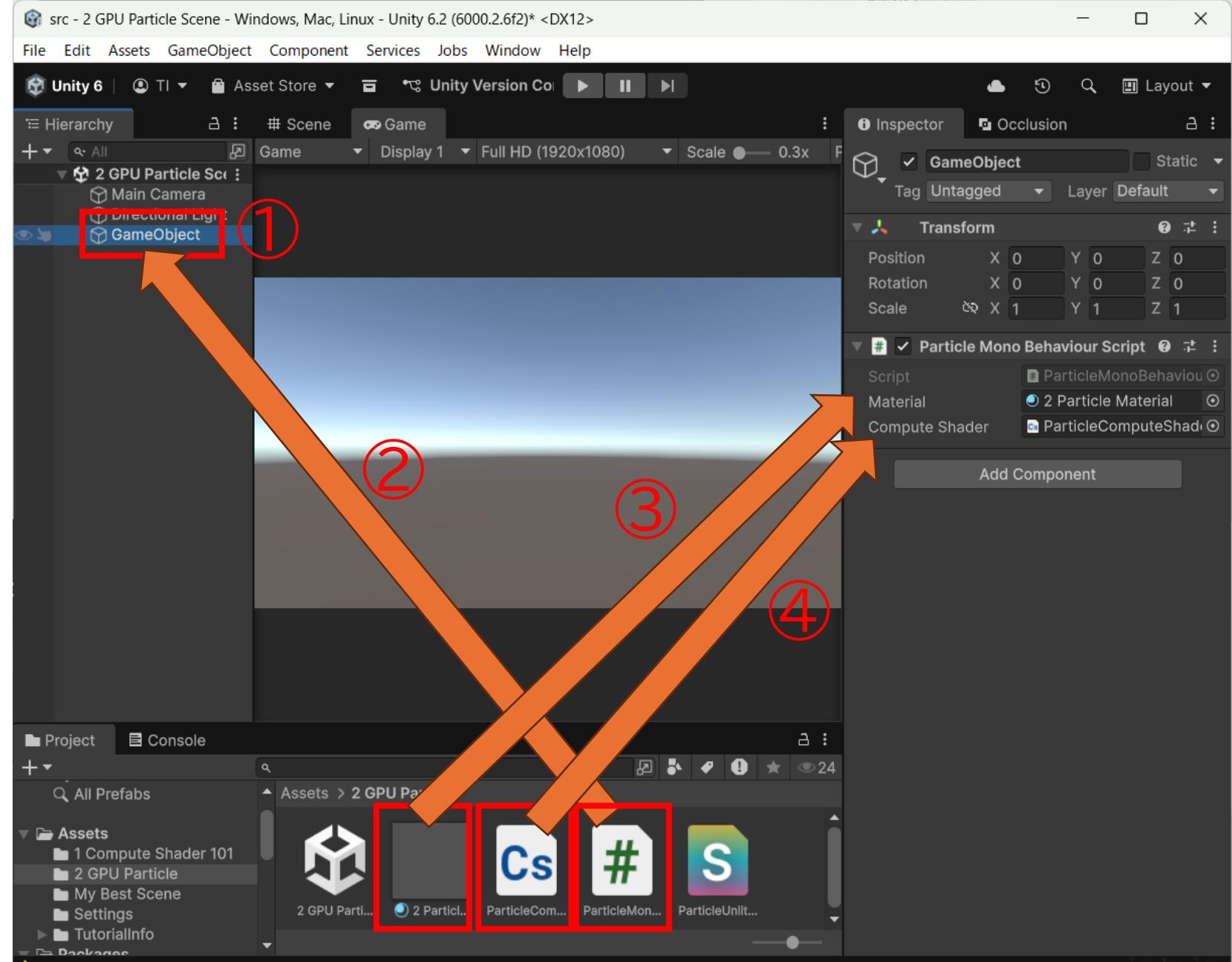
```

```
22 // オブジェクトが有効になった際に呼ばれる ParticleMonoBehaviourScript.cs
23  Unity メッセージ | 0 個の参照
24 private void OnEnable()
25 {
26     // パーティクルの情報を格納するバッファ
27     buffer = new ComputeBuffer(
28         PARTICLE_NUM,
29         Marshal.SizeOf(typeof(Particle)),
30         ComputeBufferType.Default);
31
32     // 初期化
33     // 文字列で関数を検索
34     int initKernel = computeShader.FindKernel("CSInitialize");
35     computeShader.SetBuffer(initKernel, "Particles", buffer);
36     初期化関数の実行 computeShader.Dispatch(initKernel, PARTICLE_NUM / THREAD_NUM, 1, 1);
37
38     // 更新後の設定
39     updateKernel = computeShader.FindKernel("CSUpdate");
40     computeShader.SetBuffer(updateKernel, "Particles", buffer);
41
42     // 描画用のマテリアルの設定
43     material.SetBuffer("Particles", buffer);
44 }
```

```
44 // プロジェクトが無効になった際に呼ばれる
45 ⑤Unity メッセージ|0 個の参照
46     private void OnDisable()    OnEnableの対となる解放処理
47     {
48         buffer.Release();
49     }
50
51 // 動かす
52 ⑤Unity メッセージ|0 個の参照
53     void Update()
54     {
55         computeShader.SetFloat("deltaTime", Time.deltaTime);
56         更新関数の実行 computeShader.Dispatch(updateKernel, PARTICLE_NUM / THREAD_NUM, 1, 1);
57     }
58
59 ⑤Unity メッセージ|0 個の参照
60     void OnRenderObject()
61     {
62         material.SetPass(0);
63         描画処理は直接呼び出す Graphics.DrawProceduralNow(MeshTopology.Points, PARTICLE_NUM);
64     }
65 }
```

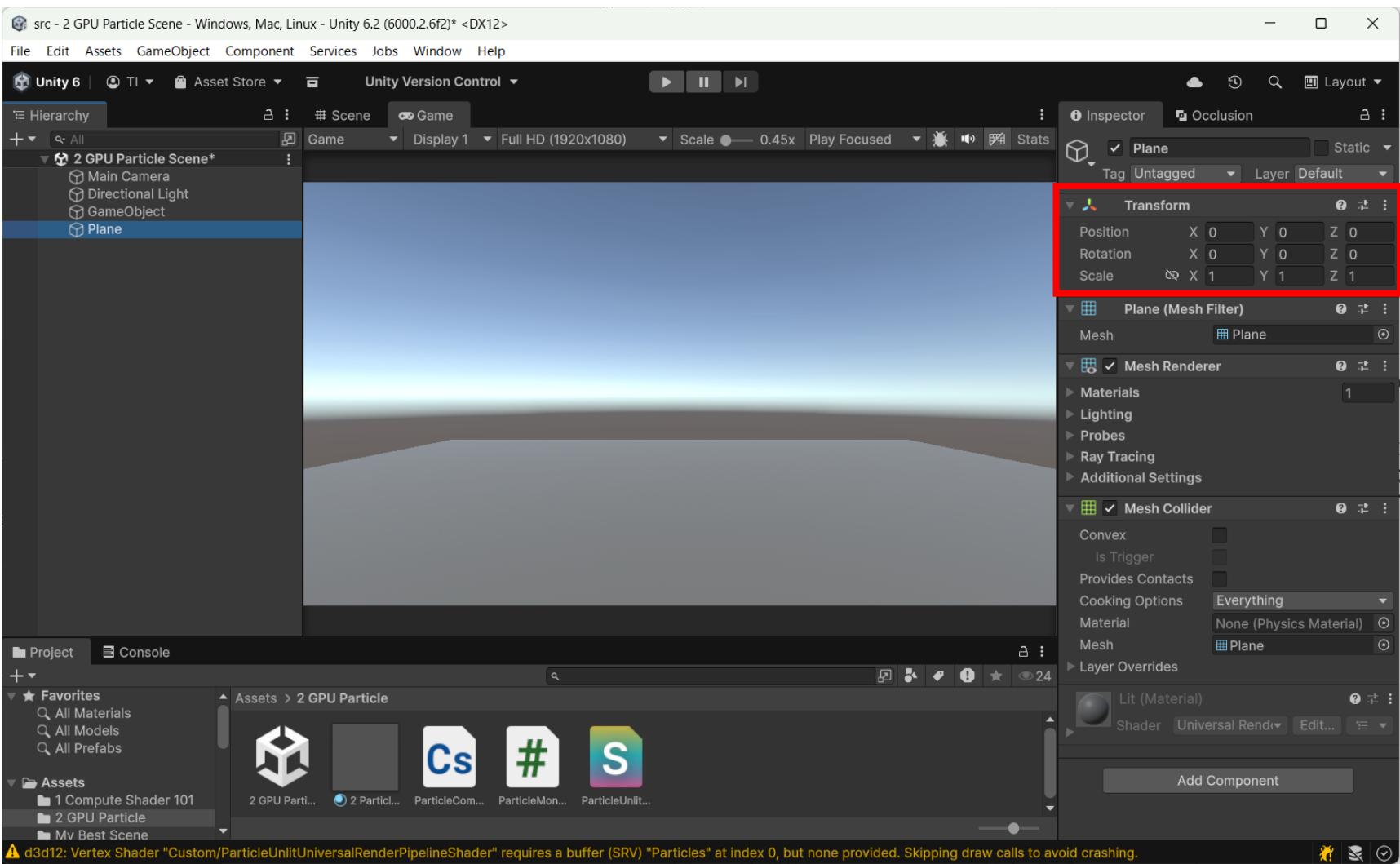
オブジェクトの追加

1. 「Create Empty」で空のオブジェクトを生成
 - ・名称例: GameObject
2. ParticleMonoBehaviourScriptをオブジェクトに追加
 - ・ParticleMonoBehaviourScriptのプロパティを設定
3. Material: 2 Particle Material
4. Compute Shader: ParticleComputeShader



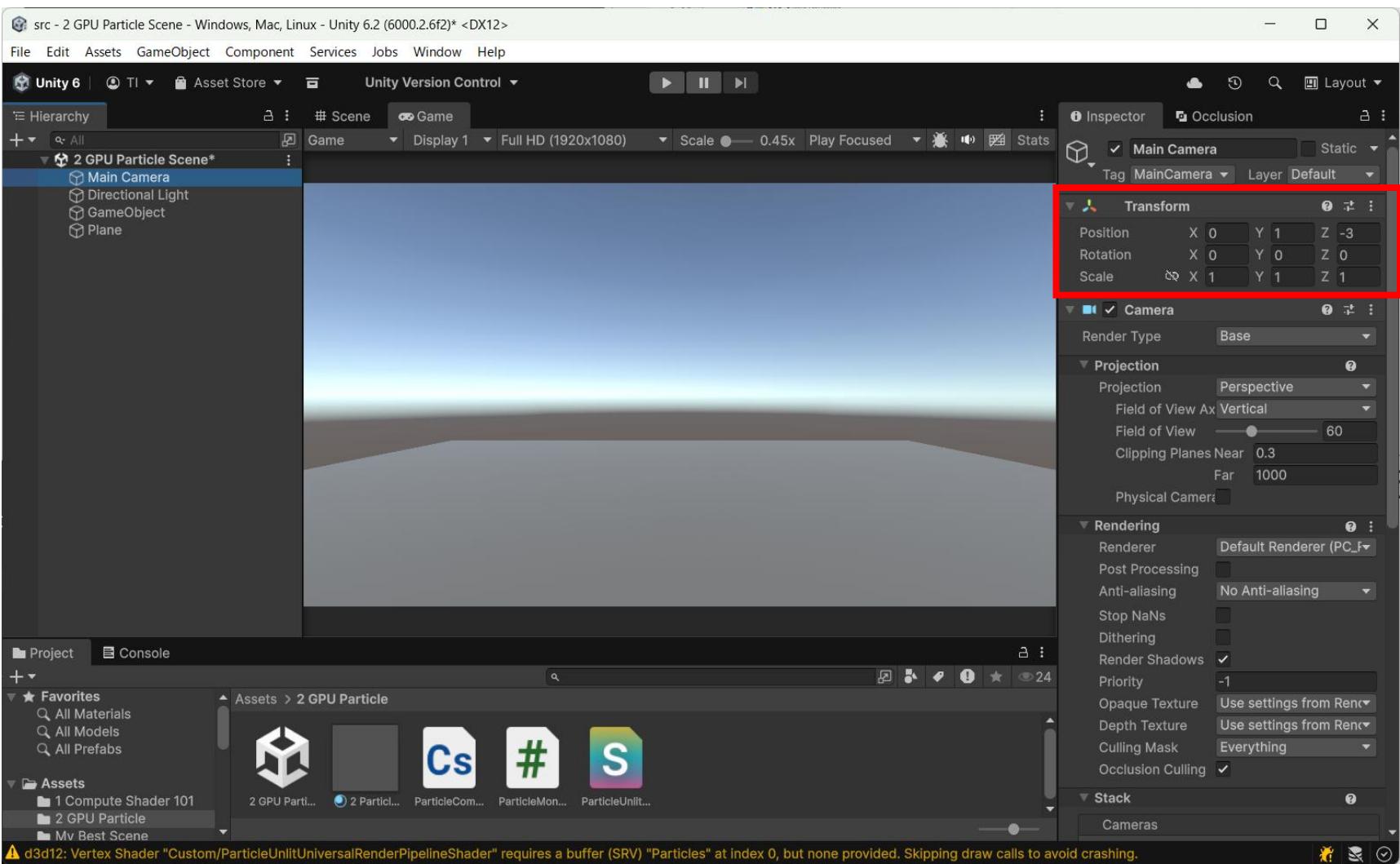
床の追加

- 何もない場所を跳ね返ると変なので床を追加
 - Planeを追加
 - 原点に配置
 - Scaleは1にしているがはみ出しているので、大きくするのも良い

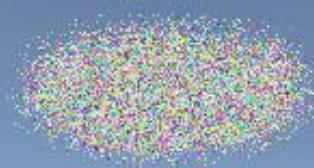


カメラ

- 見やすい位置に調整
 - ここでは-10から-3に近づけた



やってみよう



まとめ

- コンピュートシェーダ概要
- 簡単なコンピュートシェーダ
 - デフォルトで生成されるコンピュートシェーダコードを素直に表示
- GPUパーティクル
 - 大量の計算を並列に実行する