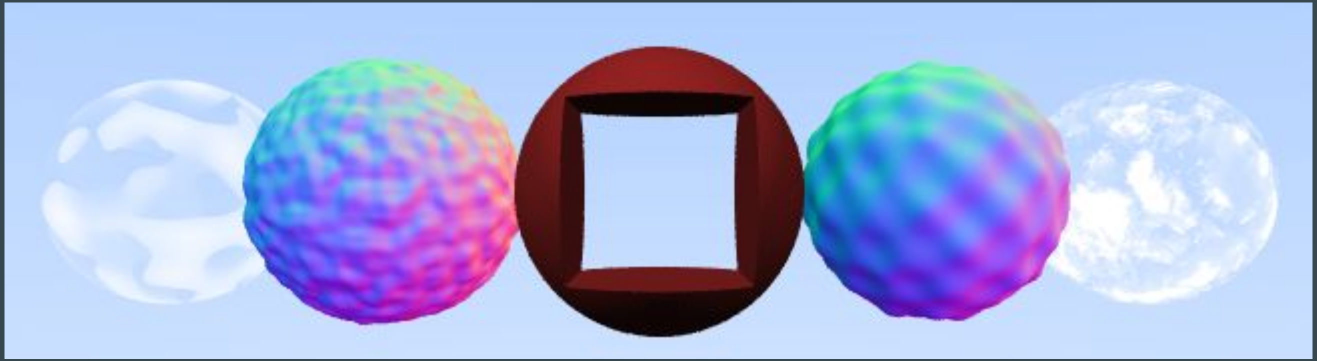
CSCI 580: 3D Graphics & Rendering
University of Southern California

Procedural Rendering w/ Ray Marching

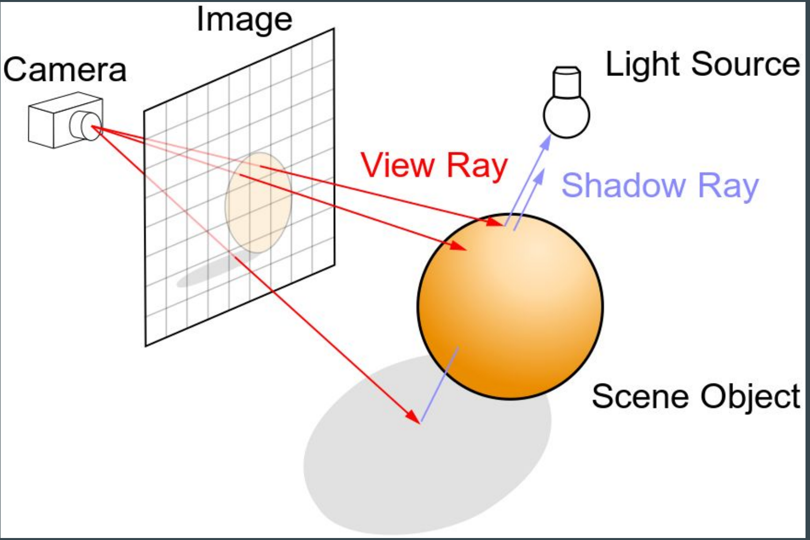


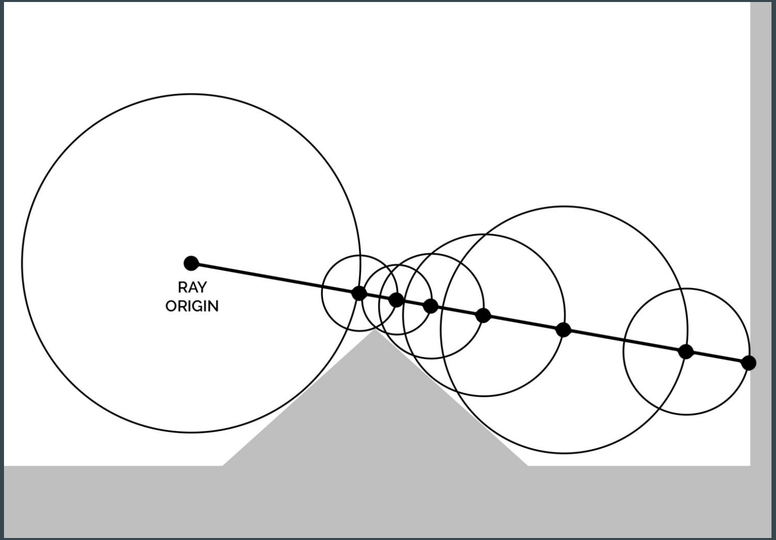
By: Christian Robles, Hoseung Lee, Samuel Yin, Vansh Dhar

Overview

- Ray Marching Intro & Core Renderer Implementation
- Constructive Solid Geometry
- Procedural Materials & Surfaces
- GPU Parallelization

Ray Tracing vs. Ray Marching





- Explicit vs. Implicit Intersection Tests

Renderer Implementation

- *Surface.* `double distance(vec3 p)`
- *Material.* `vec3 color(ray r, vec3 p, vec3 N, vector<light> lights)`
- *Scene.*

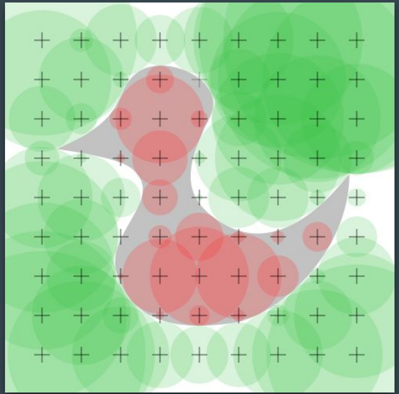
```
bool near_zero(double d)

double distance_estimator(vec3 p)

vec3 normal(vec3 p)

bool march(ray r, hit_record rec)

vec3 ray_color(ray r)
```

Marching Algorithm

```
for each pixel
    let ray R be a camera ray in the direction of the pixel
    let t be 0
    while t < t_max
        let D be the distance from R(t) to the nearest surface
        if D is near-zero
            store intersection information on the hit record
            return true
        if D < 0.001, D = 0.001
        t += D
    return false
```

Normals

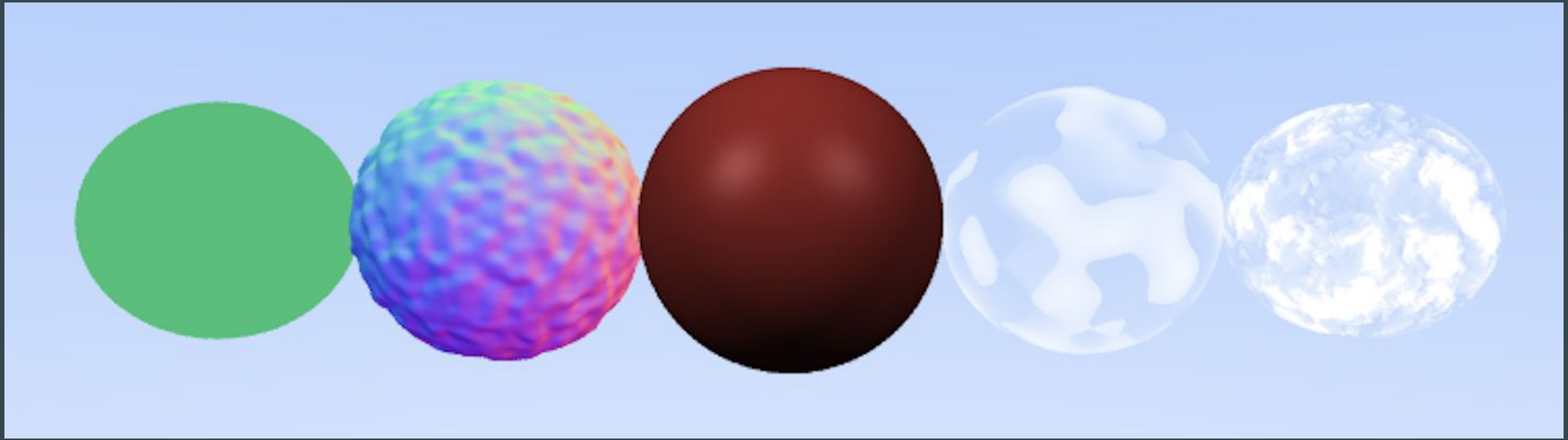
$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right)$$



$$\vec{n} = \begin{bmatrix} f(x+\varepsilon, y, z) - f(x-\varepsilon, y, z) \\ f(x, y+\varepsilon, z) - f(x, y-\varepsilon, z) \\ f(x, y, z+\varepsilon) - f(x, y, z-\varepsilon) \end{bmatrix}$$

Materials

- Flat: returns a constant color
- Normals: maps a normal vector to RGB
- Diffuse: simple diffuse material with directional lighting
- Clouds: procedural clouds with depth marching

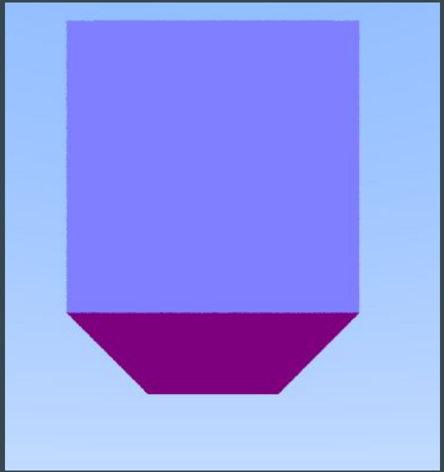


Surfaces

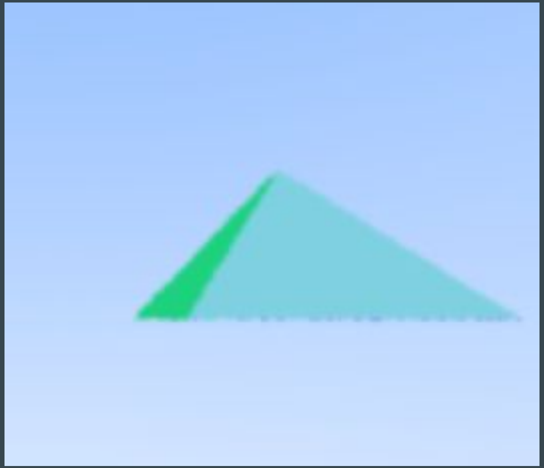
SDF for some point $p(x,y,z)$

- Sphere: |(p - C)| - r

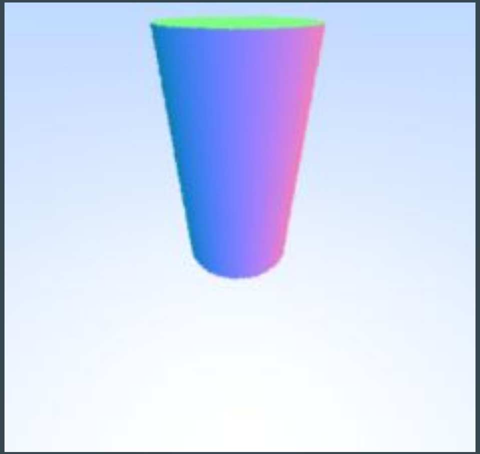
- Box: length(max(|P| - R, 0))



Surfaces - Continued



Pyramid

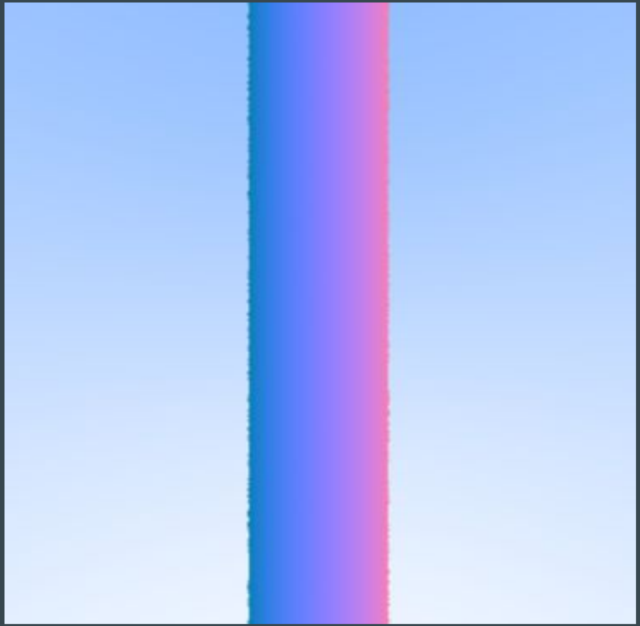


Cylinder

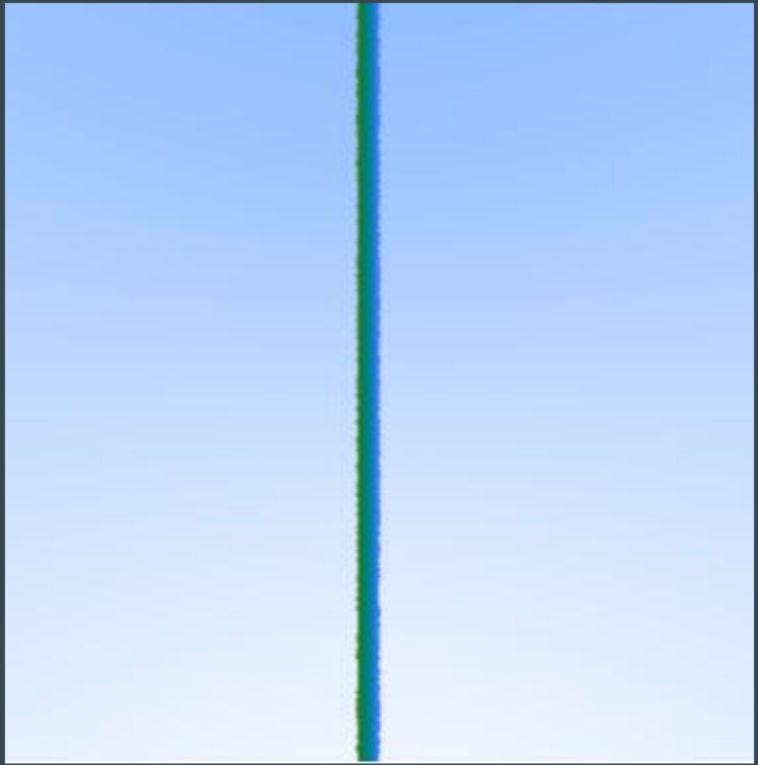
Triangular Prism

Surfaces - Continued

Infinitely Long Cylinder



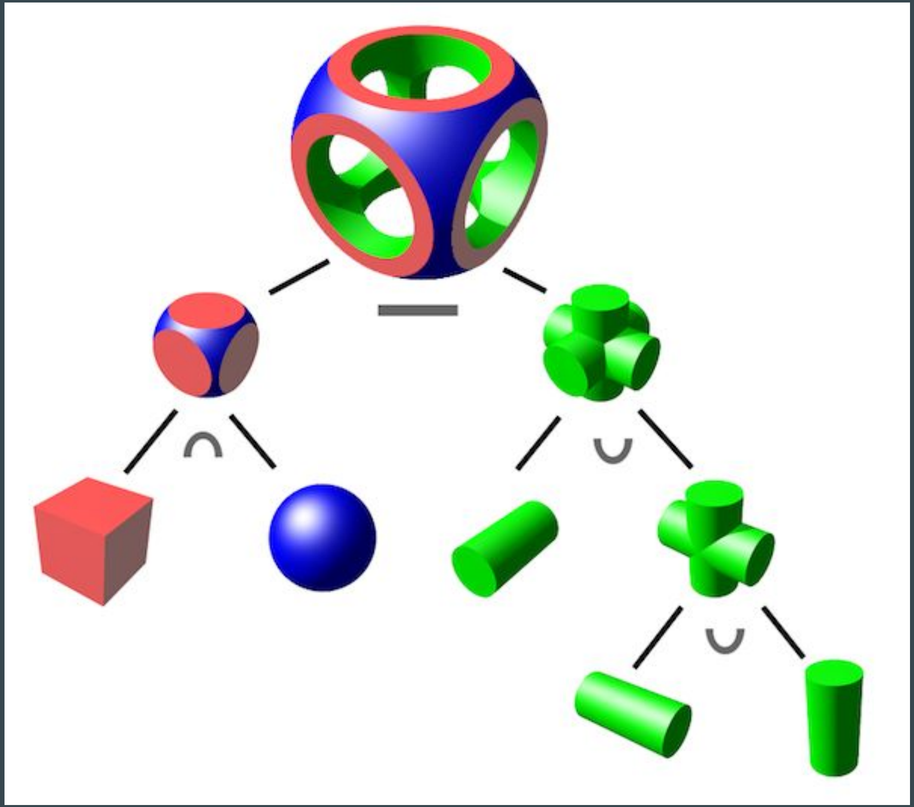
=



Constructive Solid Geometry

Constructive Solid Geometry is the process of combining simple objects using Boolean operators to create more complex objects.

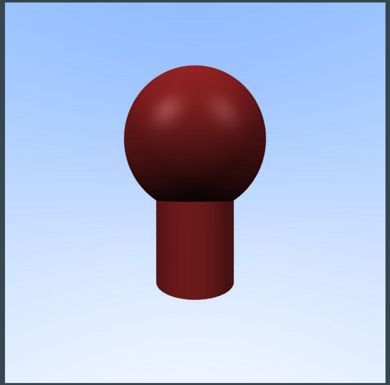
Ray Marching makes this process simple!

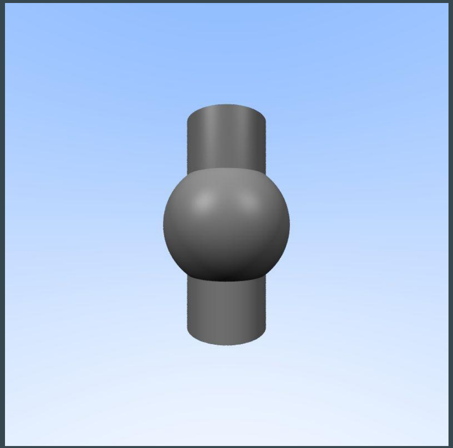


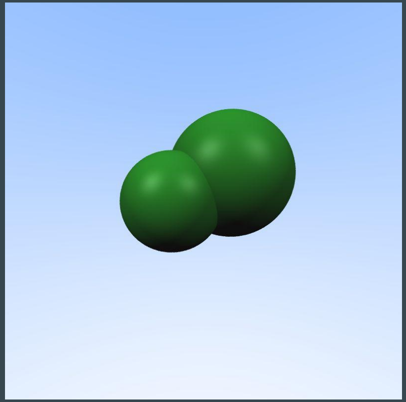
Union

Union is the min distance between objects

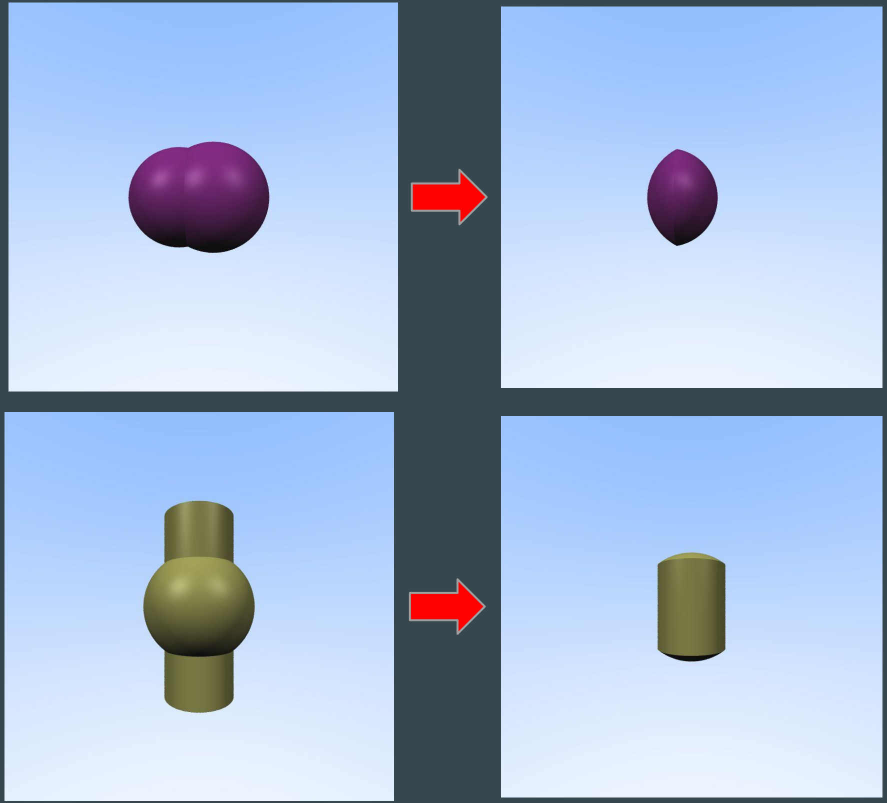
We already are finding the minimum distance in the ray marching algorithm so we combine objects by default.







Intersection



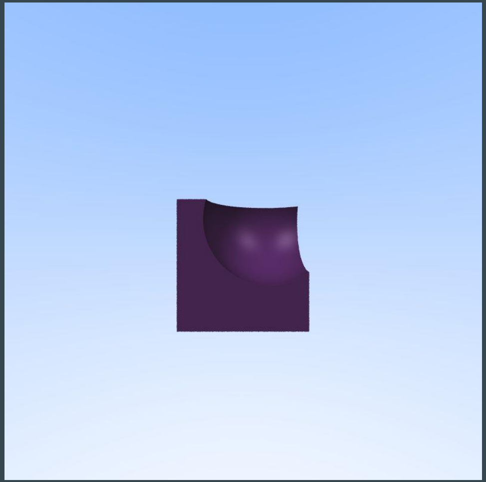
To get the intersection, we just need to get the maximum of the distances instead of the minimum

The only way the maximum of 2 distances can be close to 0 (near the surface) is if both are close to 0.

Difference

To get the difference, we negate the distance of one of the object and take the maximum distance.

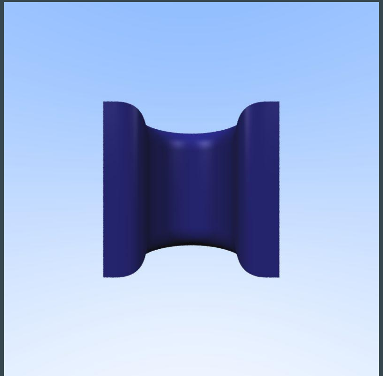
This essentially creates an object that can only exist outside the inverted object and inside the other object



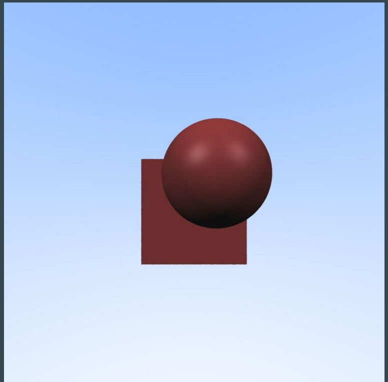
Smoothing

By applying a blending function we can smooth out the edges between objects

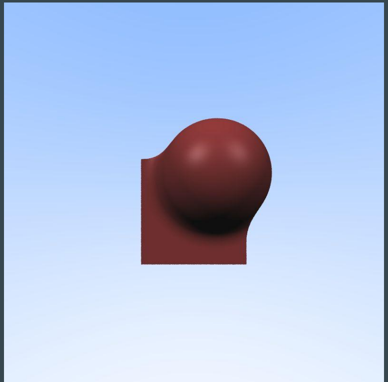
```
double smoothUnion(double d1, double d2, double k){
    float h = max(k - |d1 - d2|, 0.0);
    return min(d1, d2) - h * h * 0.25/k;
}
```



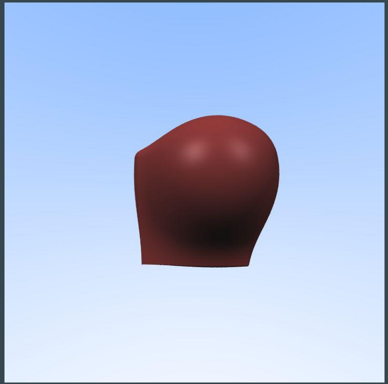
Smoothing Examples for Difference



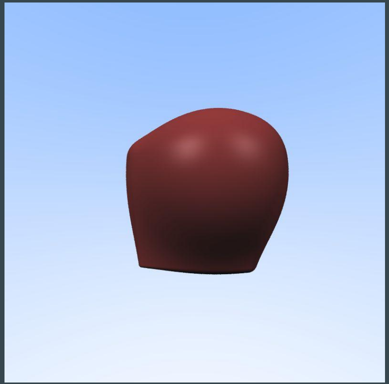
No Smoothing



K = 0.2



K =0.6



K = 0.9

Rendering Clouds

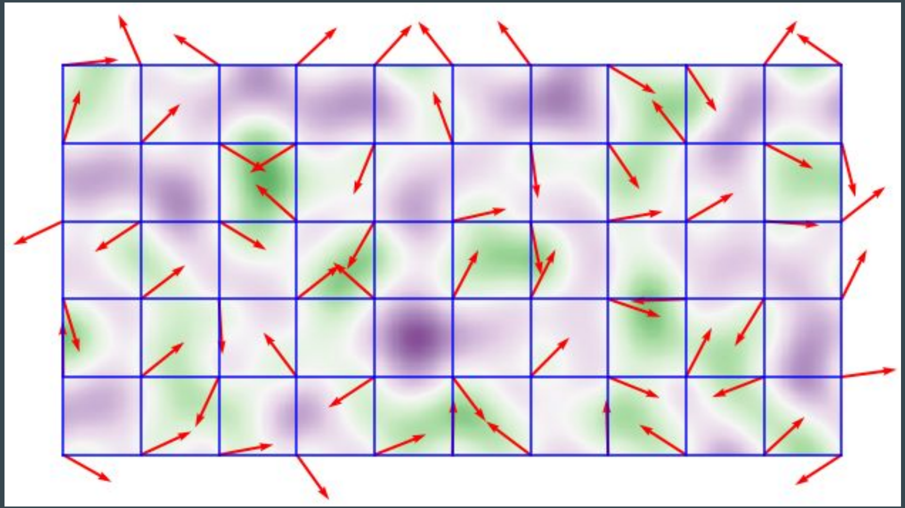
- A texture function returns a [0, 1.0] texture value *T* for some input *p(x,y,z)*
- We can additionally march through our surfaces and generate texture values in 3d

Noise Functions

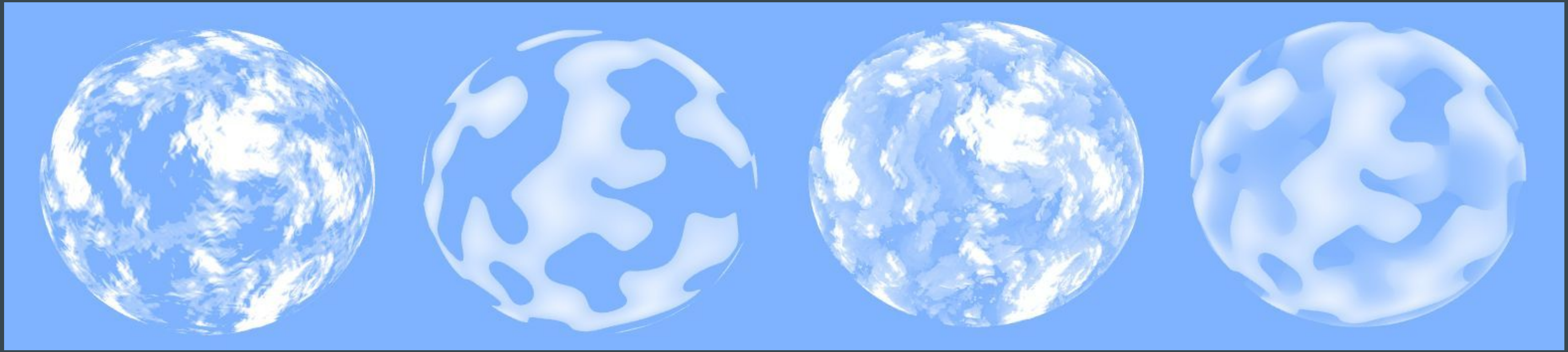
Gardner Noise

$$T(X,Y,Z) = k\sum_{i+1}^{n}[C_i \, Sin(FX_i X + PX_i) + T_o] \sum_{i=1}^{n}[C_i \, Sin(FY_i Y + PY_i) + T_o].$$

Perlin Noise

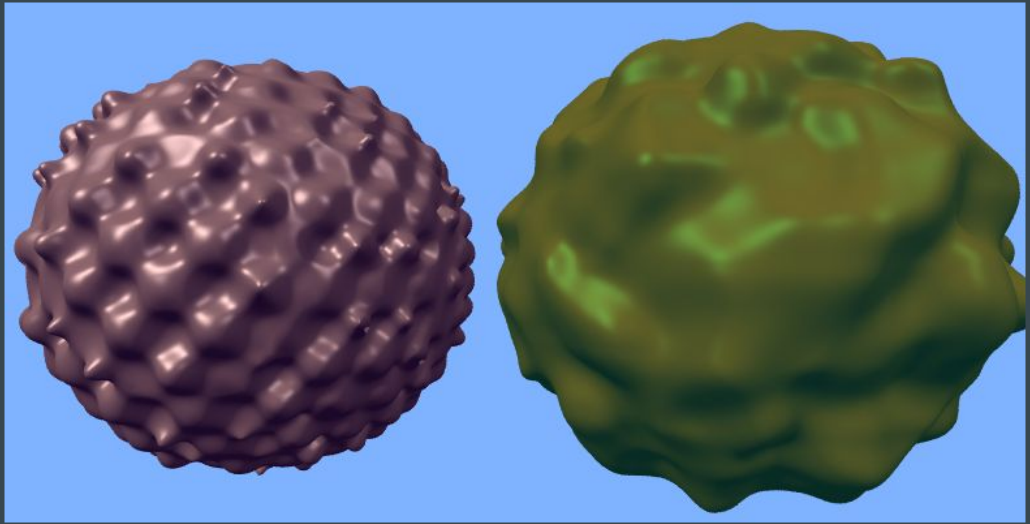


Results



- Gardner Sphere, Perlin Sphere, Depth Marched Gardner Sphere, Depth Marched Perlin Sphere
- Lighting can be applied to improve physical plausibility

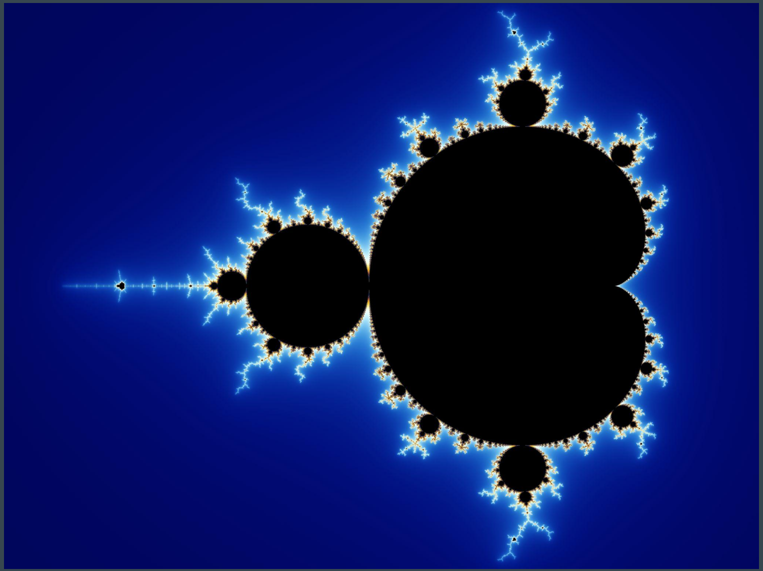
Procedural Surfaces

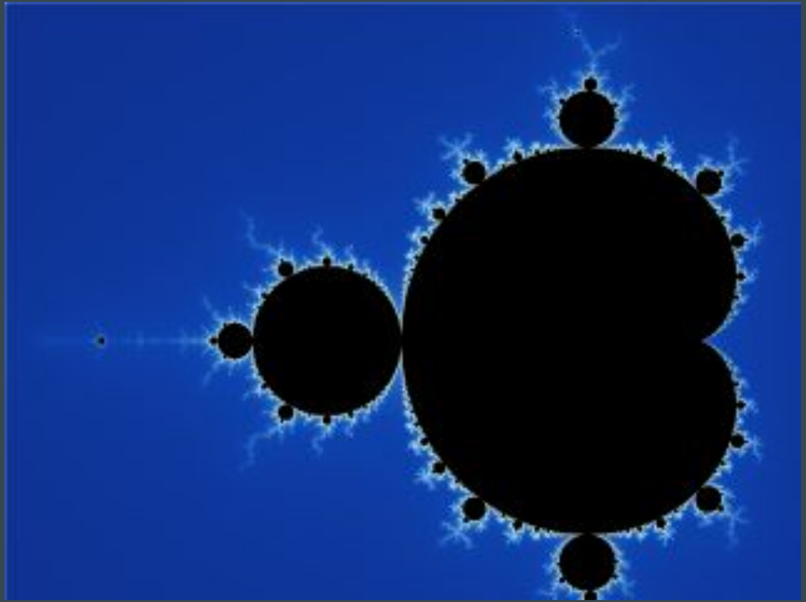


- We can modify the SDF of a surface with a procedural displacement value
- Similar to bump mapping, but alters the surface rather than texture
    - We get bump mapped textures “for free” with gradient normals

Fractals

- Fractals are infinitely complex mathematically defined structures





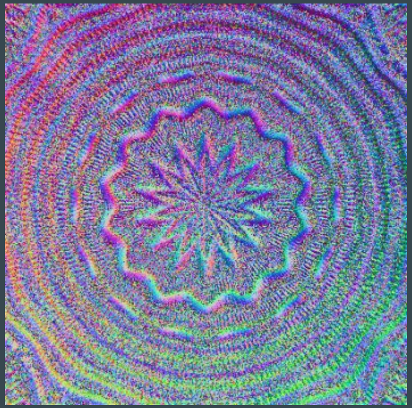
- https://en.wikipedia.org/wiki/Mandelbrot_set

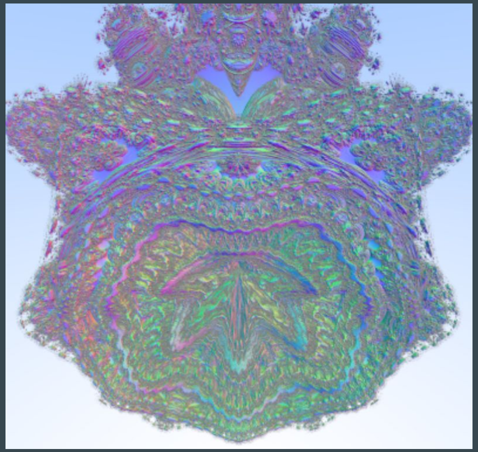
Fractals - Mandelbulb

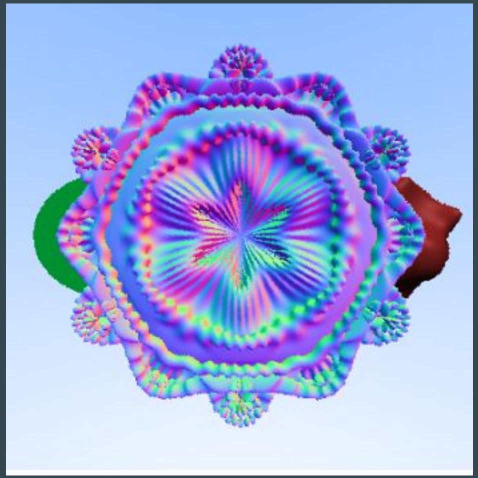
-Ray marching works well with infinitely complex shapes like fractals. Analytic intersection doesn’t exist

-No function that tells you “your ray is X units of distance away from the fractal”

-There is a function that tells you the fractal is at MOST X units of distance away







Parallelization using CUDA

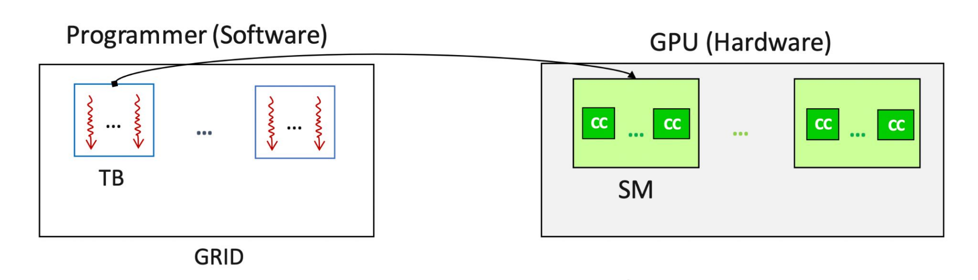


Diagram depicting software and hardware abstraction for GPU programming

Algorithm 1 : Algorithm executed by each Thread/Pixel

$i \leftarrow ThreadIdx.x + BlockDim.x * BlockIdx.x$
$j \leftarrow ThreadIdx.y + BlockDim.y * BlockIdx.y$
Ensure: $i < Image\_Width$ and $j < Image\_Height$
$N \leftarrow No\_of\_Samples$
$Color_{ij} \leftarrow 0$
while $N \neq 0$ **do**
  $i+ = Random(-1, 1)$
  $j+ = Random(-1, 1)$
  $ray = Compute\_Ray(Camera\_Info, i, j)$
  $Color_{ij}+ = Compute\_Color(3D\_Scene, ray)$
  $N \leftarrow N - 1$
end while
$FrameBuffer[j, i] = Color_{ij} / No\_of\_Samples$

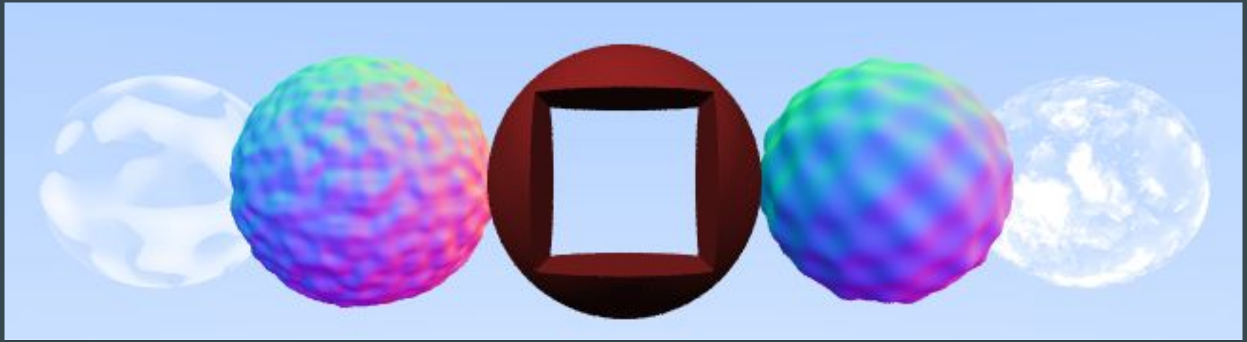
The Algorithm run by each Thread

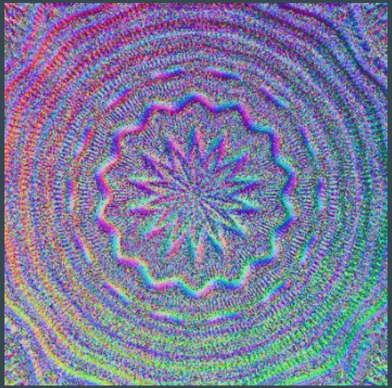
Parallelization Experiments and Results

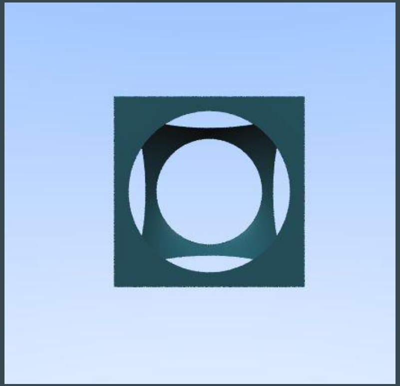
Resolution	Block Size	Number of Samples	Parallel Time	Serial Time	Speed Up	Branch Efficiency	A/T Occupancy
256x256	8x8	8	0.027s	29.483s	1091.963	99.27	83.42/87.5
256x256	16x16	8	0.031s	29.483s	951.065	99.27	80.14/87.5
256x256	32x8	8	0.042s	29.483s	701.976	97.54	76.59/83.33
256x256	128x1	8	0.038s	29.483s	775.868	99.27	77.37/83.33
256x256	256x1	8	0.033s	29.483s	893.424	98.65	79.28/87.5
512x512	8x8	8	0.073s	117.817s	1613.932	99.38	82.74/87.5
512x512	16x16	8	0.081s	117.817s	1454.531	99.27	78.94/87.5
512x512	32x8	8	0.096s	117.817s	1227.260	99.27	72.41/83.33
512x512	128x1	8	0.092s	117.817s	1280.620	98.65	74.53/87.5
512x512	256x1	8	0.093s	117.817s	1266.849	98.65	74.68/87.5
1024x1024	8x8	8	0.191s	476.936s	2497.047	99.38	83.65/87.5
1024x1024	16x16	8	0.209s	476.936s	2281.990	98.65	81.12/87.5
1024x1024	32x8	8	0.243s	476.936s	1962.700	99.38	74.64/83.33
1024x1024	128x1	8	0.241s	476.936s	1978.988	98.65	75.23/83.33
1024x1024	256x1	8	0.244s	476.936s	1954.656	98.65	75.87/83.33

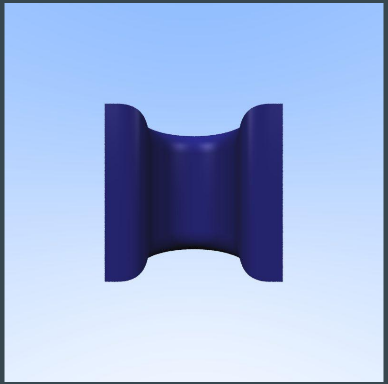
Table 1: In the above table, we list the different parameter configurations of our CUDA program and the SpeedUp achieved for those configurations. We also list the Branch Efficiency and Achieved/Theoretical Occupancy for each configuration.

Conclusion, Q&A









References

- https://raytracing.github.io/
- Geoffrey Y. Gardner. Visual Simulation of Clouds. SIGGRAPH '85
- Ken Perlin. 1985. An Image Synthesizer. SIGGRAPH '85
- https://developer.nvidia.com/blog/accelerated-ray-tracing-cuda/
- https://www.skytopia.com/project/fractal/mandelbulb.html
- https://iquilezles.org/articles/distfunctions/
- http://jamie-wong.com/2016/07/15/ray-marching-signed-distance-functions/
- https://michaelwalczyk.com/blog-ray-marching.html