Mandelbulb Visualizer With Raymarching Demo

Riley Mahr, Alex Stiyer

The Problem:

Implement a Ray Marcher in Unity Compute Shaders to Render a Mandelbulb. Include interactive Ray Marching demos for learning purposes.

What is a Shader?

- GPU accelerated script that acts on all pixels of a view or all vertices of a shape

- Entire shader script is ran for each Pixel or Vertex in parallel making this method of computer graphics extremely efficient

- Can take advantage of dedicated logic units of GPU for matrix math, helper functions, and other graphics related calculations

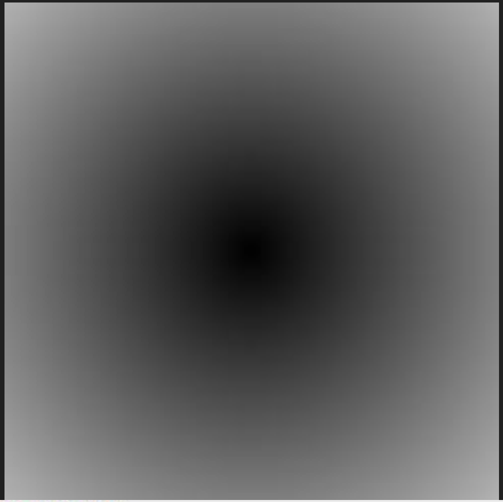
Challenges of Using a Shader

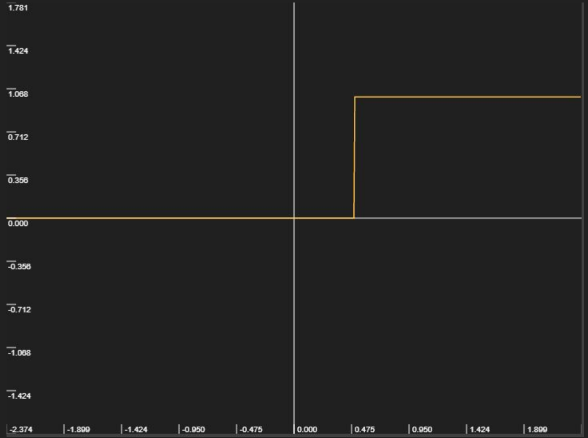
- Because you are working on a pixel/pixel level, creating shapes relies on stateless functions based on pixels.xy positions

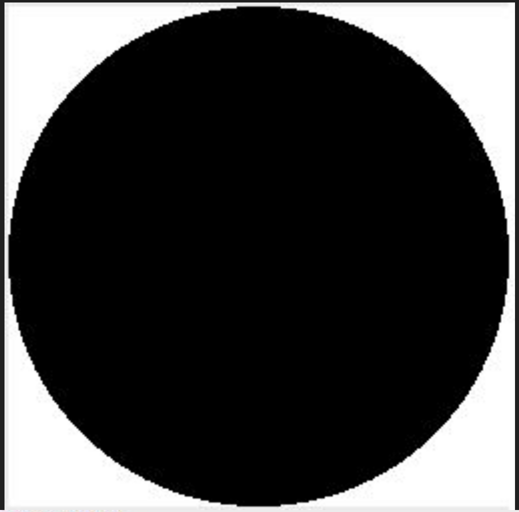
- In 2D you must use **Distance Fields** and **Shaping Functions** to create 2D shapes

- In 3D we use **Signed Distance Functions** to represent Volumes

Example of Shaping Function with Distance Field







```
void main(){
    vec2 st = gl_FragCoord.xy/u_resolution;

    float pct = 0.0;

    pct = distance(st,vec2(0.5));
    vec3 color = vec3(pct);
    gl_FragColor = vec4( color, 1.0 );
}
```

f(x) = step(.5,x)

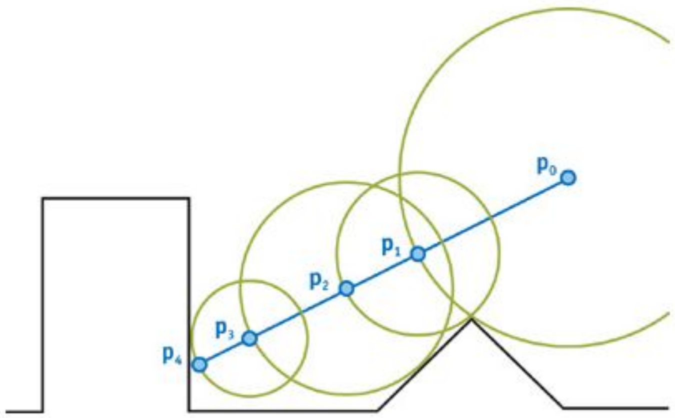
vec3(pct) = (pct,pct,pct)
0,0,0 or 1,1,1

```
void main(){
    vec2 st = gl_FragCoord.xy/u_resolution;

    float pct = 0.0;

    pct = distance(st,vec2(0.5));
    vec3 color = step(.5,vec3(pct));
    gl_FragColor = vec4( color, 1.0 );
}
```

What is Ray Marching?

Raymarching is a rendering technique similar to ray tracing

Raymarching uses Signed Distance Functions to represent 3D volumes



From GPU Gems 2: Chapter 8.

Algorithm Steps

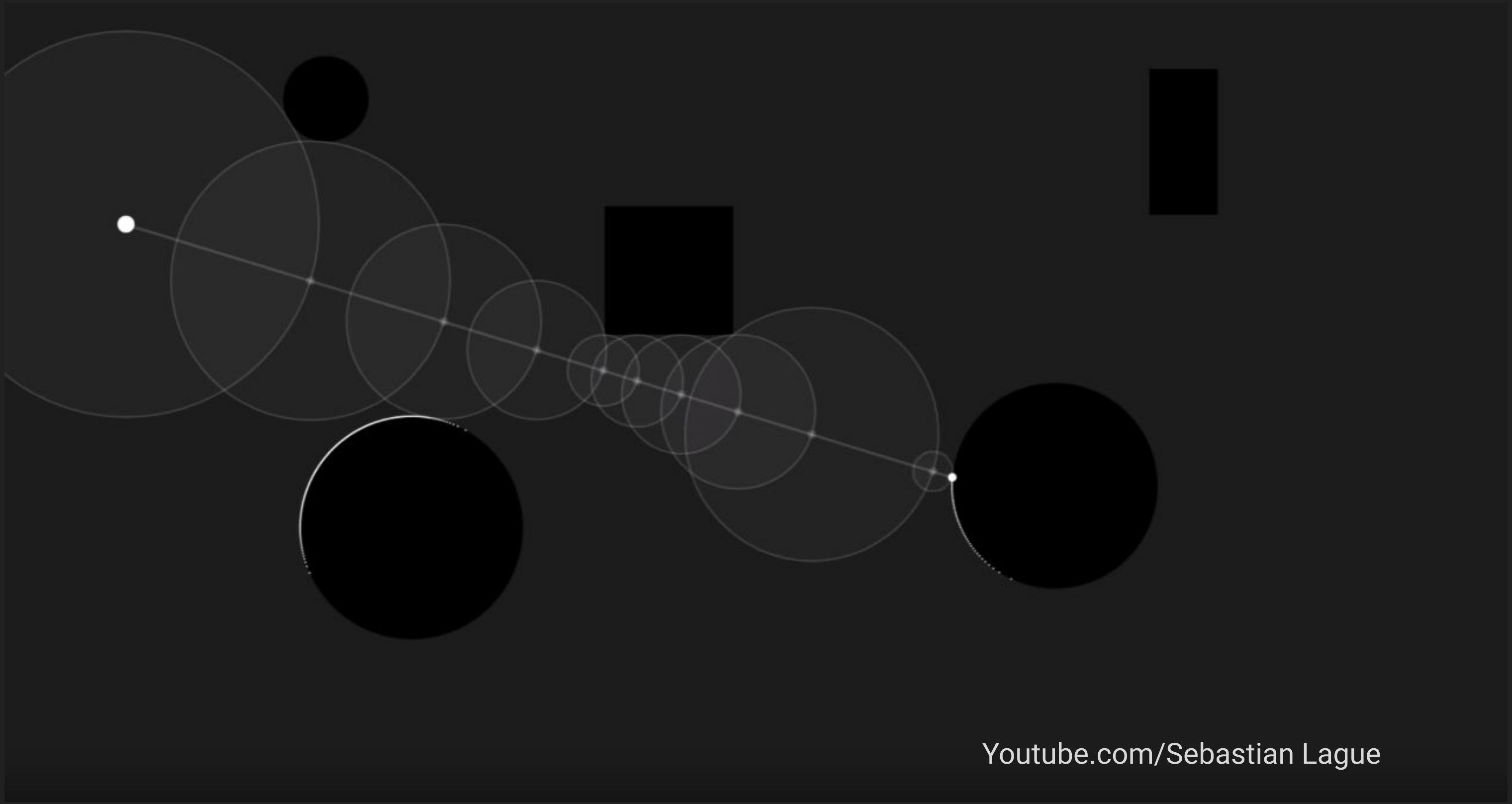
For each ray of the camera, given its direction and origin:

1) Marching_Point = origin

2) Calculate Distance to Surface (in any direction)
   a) Guarantees we will not choose point inside surface
   b) We will use SDF to represent surfaces

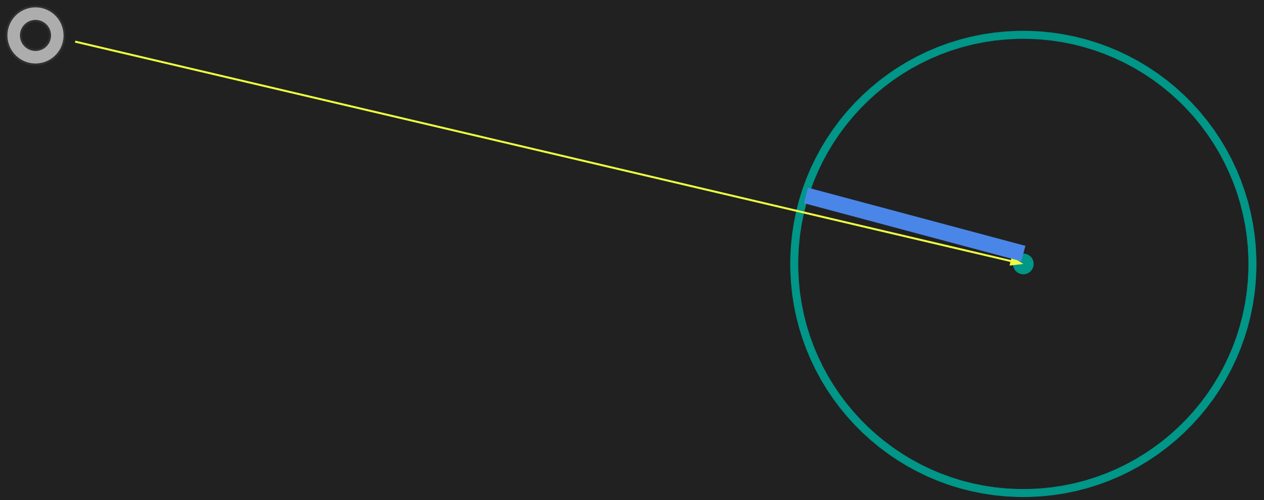
3) Move Marching_Point to calculated distance in direction opposite of camera

4) Repeat until the distance to surface is very small, we consider this a hit


Youtube.com/Sebastian Lague

Example SDF (sphere)

```
float DistanceToSphere(Ray ray, Sphere sphere) {
    return length(ray.origin - sphere.origin) - sphere.radius;
}
```



Youtube.com/TheArtOfCode

Main Loop:

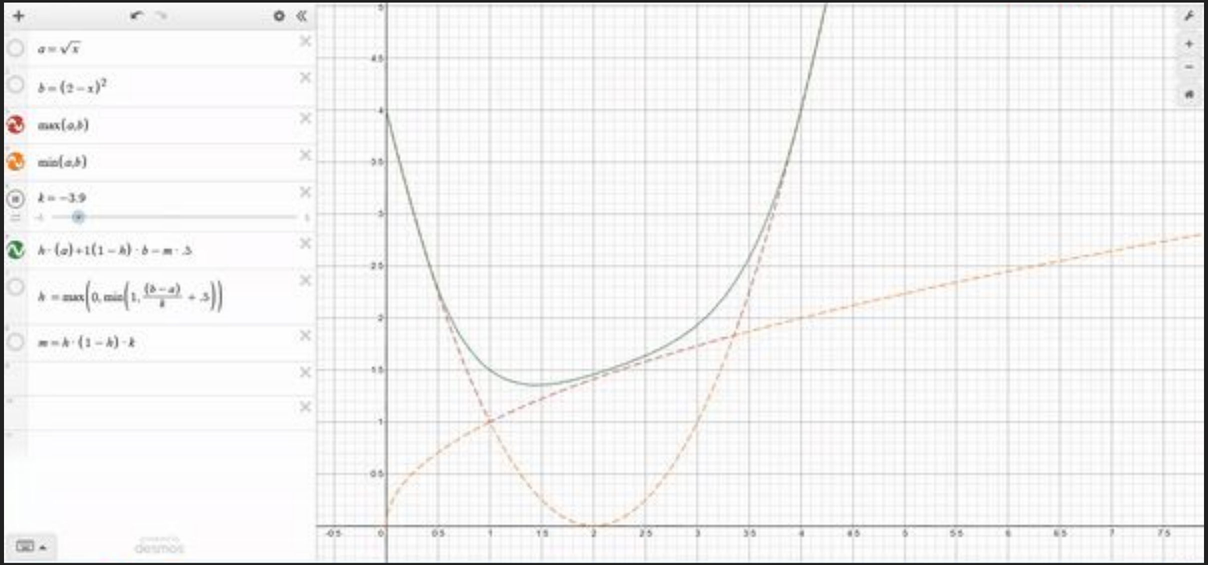
```
Scene scene = setScene();

uint width, height;
Result.GetDimensions(width, height);

float2 uv = float2((id.xy + float2(0.5f, 0.5f)) / float2(width, height) * 2.0f - 1.0f); //normalize coordinates to the window size

Ray ray = CreateCameraRay(uv);
float rayDistance = 0.0f; //March point's distance from the camera
int marchingSteps = 0; //Number of march points on ray

float maxDistance = 100.0f; //Effectively the "Far Plane"
int maxSteps = 250; //How many march operations we are willing to compute
float surfaceThreshold = 0.001f; //What distance from surface (march distance) we consider ahit

float4 result = float4(ray.direction * 0.5f + 0.5f, 1.0f); // This will be background gradient for now

while(rayDistance < maxDistance && marchingSteps < maxSteps){ //While we haven't surpassed maximum distance or steps
    marchingSteps += 1; //increment number of march steps we have taken
    float sceneDistance = DistanceToScene(ray, scene); //get the distance to the scene

    if(sceneDistance < surfaceThreshold){ //When the distance is less than the surface threshold, we consider this a hit on the scene
        result = float4(float(0.1 * marchingSteps), 1.0f, 1.0f, 1.0f);
        break;
    }
    //when we don't get a hit we march the ray equal to the distance to scene, move ray position for SDF calculation
    //increase distance from camera.
    ray.position += ray.direction * sceneDistance;
    rayDistance += sceneDistance;
}

Result[id.xy] = result;
```

DistanceToScene

```
struct Shape
{
    int type; // 0 = sphere, 1 = cube;
    float3 origin;
    float3 size;
};

struct Scene {
    Shape A;
    Shape B;
    Shape shapes[10];
    int numShapes;
    int mode; //0 none, 1 merge, 2 cut, 3 clip
};
```

```
float DistanceToScene(Ray ray, Scene scene){
    Shape A = scene.A;
    Shape B = scene.B;
    A.origin.x *= 2*_SinTime.z;

    //calculate distance from scene for both shapes
    float dA;
    if(A.type == 0){
      dA =  sphereSDF(ray, A);
    }else{
       dA = boxSDF(ray, A);
    }

    float dB;
    if(B.type == 0){
       dB =  sphereSDF(ray, B);
    }else{
       dB = boxSDF(ray, B);
    }

   //operation to perform on SDF values
   switch (scene.mode){
      case 0: //normal
        return min(dA,dB);
        break;

      case 1: //blend
        return smin(dA,dB,1);
        break;

      case 2: //clip
        return max(dA,dB);
        break;

      case 3: //cut shape B
        return max(dA,dB*-1.0);
        break;

      case 4: //cut shape A
        return max(dB,dA*-1.0);
        break;
   }

   return -1;

}
```

Smooth Min Function




```
float smin( float a, float b, float k )
{
    float h = max( k-abs(a-b), 0.0 )/k;
    return min( a, b ) - h*h*k*(1.0/4.0);
}
```

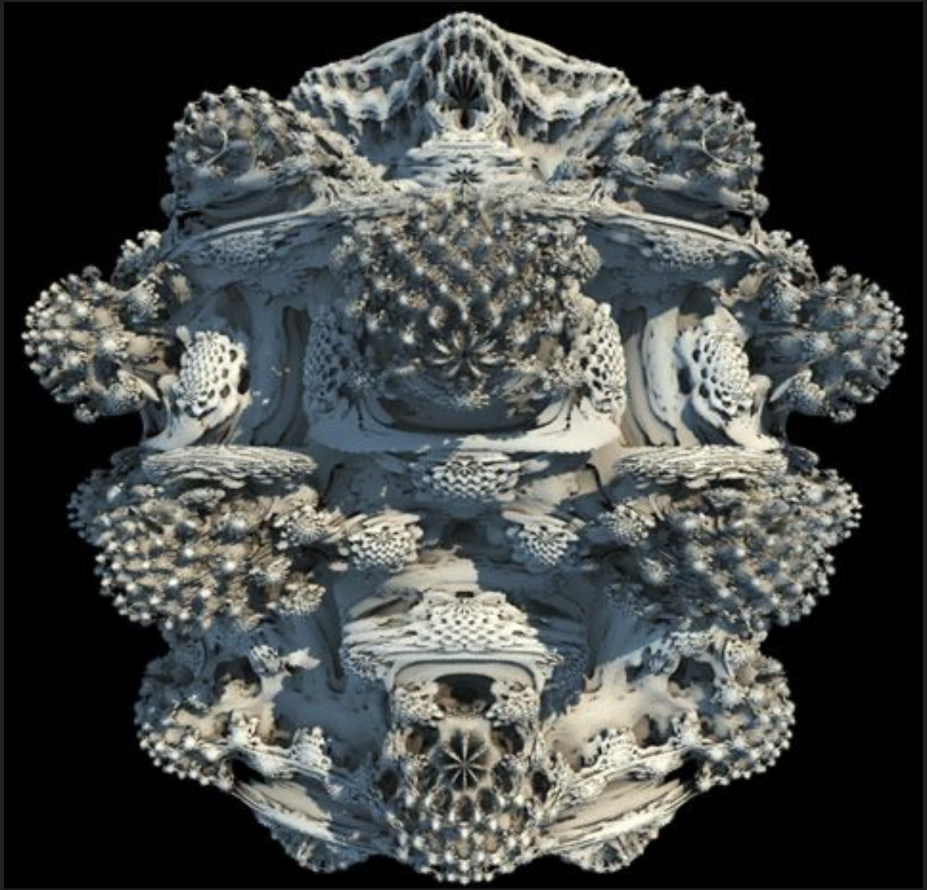
Polynomial Method from `Inigo Quilez`

3D Fractal

Expansion of MandelBrot set into 3D

```
// extract polar coordinates
    float wr = sqrt(dot(w,w));
    float wo = acos(w.y/wr);
    float wi = atan(w.x,w.z);

    // scale and rotate the point
    wr = pow( wr, 8.0 );"
    wo = wo * 8.0;"
    wi = wi * 8.0;"

    // convert back to cartesian coordinates
    w.x = wr * sin(wo)*sin(wi);
    w.y = wr * cos(wo);
    w.z = wr * sin(wo)*cos(wi);
```

https://www.iquilezles.org/www/articles/mandelbulb/mandelbulb.htm



3D Fractal

```
// Derived from https://www.iquilezles.org/www/articles/mandelbulb/mandelbulb.htm
float2 mandelbulbInfo(float3 pos){

    float3 p = pos;
    float dr = 1.0;
    float r = 0.0;
    float power = _Power;
    //float power = 8; // this will affect complexity of the shape and we should be able to change this
    int iterations = 0;

    for (int i = 0; i < 15; i++){
        iterations = i;

        r = length(p);

        if(r>2){
            break;
        }

        //convert to polar coordinates
        dr = pow(r, power - 1.0) * power * dr + 1.0;
        float theta = acos(p.z/r);
        float phi = atan2(p.y,p.x);

        //scale and rotate the point
        float zr = pow(r,power);
        theta = theta*power;
        phi = phi*power;

        //convert back to cartesian coordinates
        float x = sin(theta)*cos(phi);
        float y = sin(phi)*sin(theta);
        float z = cos(theta);

        p = zr*float3(x, y, z);
        p+=pos;
    }
    float dst = 0.5*log(r)*r/dr;
    return float2(iterations, dst * 1);
}
```

3D Fractal

- Accomplished in same manner as SDF raymarching using the previous function.
- Demonstrates strong use case for raymarching when volumetric expressions are focus of rendering.
- An example of 3d fractals can be seen in Big Hero 6, when the main characters go through a portal.
- Many implementations can be found on websites like shadertoy.

https://www.iquilezles.org/www/articles/mandelbulb/mandelbulb.htm

Unity 3D

- For this project, Unity 3D, a popular game engine was used. Shader code was custom written, but the use of the engine allows for:
    - Compilation of shader code for multiple targets/platforms (Metal, Vulkan, Directx, etc.)
    - Easy set up of shader code
    - Built in handling of windowing systems and computer recognition of graphics pipelines (for example, GeForce experience recognizes the build on Windows 10)
    - Use of a physically accurate camera and the necessary matrices for creating a ray marcher (or ray tracer)
    - Handling of unimportant items like interface, help menus, and key detection so efforts could be focused elsewhere.
    - Easy adjustment and saving of user defined parameters.

https://unity.com

Unity Compute Shaders

- “Compute shaders are programs that run on the graphics card, outside of the normal rendering pipeline. They can be used for massively parallel GPGPU algorithms, or to accelerate parts of game rendering.”
- Uses HLSL (High Level Shading Language), similar to C
- Takes advantage of modern hardware and can have user defined sets of inputs and outputs for multiple purposes
- Platform compatibility defined in Unity Docs, but shaders are able to be used on the majority of modern platforms.

https://docs.unity3d.com/Manual/class-ComputeShader.html

Demo

- https://github.com/rilmar/RaymarcherCompute
- Uses Unity version 2019.2.12f

Resources to Learn More:

- TheBookOfShaders.com
  - Comprehensive guide to fragment Shaders (2D)
  - Being written in chunks, currently on Generative Designs Chapter
  - 3D Graphics to come later!

- ShaderToy.com
  - Create your own shader online
  - Browse other creations and discuss with community