The University of Texas at Dallas

Erik Jonsson School of Engineering and Computer Science

CS 6366: Computer Graphics

PROJECT #

Rendering the world with two triangles

Team Member:

Venkata Sai Kumar Guntupalli -VXG210063

Bhavya Thadiboina – BXT220009

Problem Summary:

Here's the issue: I'm trying to render a demo scene using the ray marching method, and I'm having trouble. In order to give the screen a wavelike effect, the demo scene aims to accomplish a smooth transition between geometric objects (Sphere, cube, and Torus). As a result, we used a method known as "Distance Aided Ray Marching" to estimate the distance to the closest surface at any given position in space. The "Signed Distance Function" can be derived for simple geometric objects like cubes and spheres using the aforementioned methods. We can create extrusion and mixing effects by modifying these SDF.

Description of work:

We have Implemented out demo scene in shadertoy IDE. My user name is Gvskumar.
We use GLSL language.
Users can develop pixel shaders with the aid of the website Shadertoy. A pixel shader determines the color of a single pixel based on the surface where the ray from the camera intersects that pixel. Shadertoy uses the WebGL API to use the GPU to render graphics in the browser.
Because the shader makes use of the GPU, each pixel may be processed in parallel.

Algorithms Implemented:
Signed Distance Fields (SDFs):
Signed Distance Fields (SDFs) are a potent method used in computer graphics to effectively and compactly depict geometric forms. A SDF provides a distance function that receives as input a 3D point and outputs the signed distance to the closest surface of the shape being represented.
The signed distance is zero if the point is on the shape's surface, negative if it is inside the shape, and positive if it is outside it. By only evaluating the distance function at each point along a ray or at a specific location in space, we are able to carry out operations like ray marching and collision detection.
SDFs' ability to represent complicated shapes using comparatively straightforward mathematical expressions is one of their key features.
Ray marching:
To render three-dimensional scenes in computer graphics, ray marching is a technique. The method involves tracing camera rays through each pixel on the screen, marching along the ray in tiny steps, and assessing the picture at each stop until the ray intersects an item or reaches its maximum length.
Signed distance functions (SDFs) and ray marching are frequently employed together to depict scene geometry. The SDF is used to calculate the signed distance to the closest item in the scene at each step along the ray. The ray is assumed to have impacted an object if the distance is less than a specified limit, and the related color and shading information is calculated using the intersection.

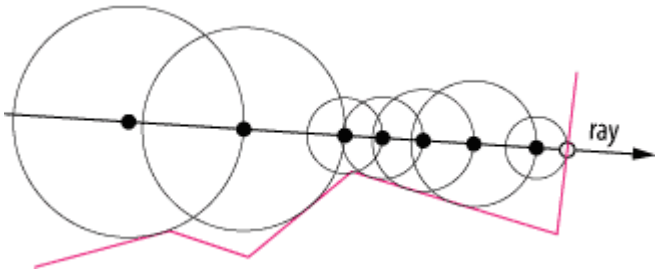

Figure 1: Ray-Marching

Normals:

As they control how light interacts with surfaces in a scene, normals are essential for lighting in computer graphics. In ray marching, we determine the normal at each point in space using the analytical derivatives of the signed distance function.
General steps to calculate the normal using SDF's:

- Use the distance function to determine the SDF value at the current location in space.
- SDF values should be calculated at locations that are somewhat off in the x, y, and z directions from the current location. The SDF value can be updated at the new point by first adding a little epsilon value to each component of the existing point.
- For each direction (x, y, and z), deduct the SDF value at the current position from the SDF value at the offset points. You will then be given the SDF's analytical derivative along each axis.
- To get the surface normal at the present position, normalize the resulting vector. To accomplish this, divide the vector's length.
- Lighting and other surface interactions can then be calculated using the resulting normal vector.

Texture:

To add texture to a distance field rendering, you can use the distance value returned by your distance function as a lookup into a texture. This works by mapping the distance value to a normalized coordinate in the texture, then looking up the color value at that coordinate.

Analysis of work done:

mapSphere(vec3 p), mapCube(vec3 p), myTorus(vec3 p, float r, float t):

In this code, the SDF for the sphere, is defined by the mapSphere(), mapCube(), myTorus() function, which takes a 3D point as input and returns the signed distance to the surface of the sphere, cube, torus at that point.

The texture is applied to the surface of the sphere, cube, torus by mapping the signed distance to a normalized texture coordinate (in the range [0,1]), using the vec2() function. This texture coordinate is then used to look up the color value in the texture, using the texture() function.

The color value is then converted to a grayscale value, using the formula (r+g+b)/3, where r, g, and b are the red, green, and blue components of the color value. This grayscale value is then combined with the signed distance, using a scaling factor, to produce a new signed distance value that takes into account the texture.

This new signed distance value is returned by the mapSphere(), mapCube(), myTorus() function, and is used in the ray marching algorithm to generate the final image.

softShadow():
This function commonly used in raymarching shaders to calculate soft shadows. It takes in the ray origin ro, light position lp, and a shadow softness factor k, and returns a float value representing the amount of shadow at the given point**.**

Shape Transition:
We gradually converted a sphere into a cube and then into a torus using the mix(x, y, a) function of GLSL. The value from x to y is linearly interpolated by this function using function 'a'.
The mix(x, y, a) function uses a to weight to achieve a linear interpolation between x and y. among them. x(1 a) + y(a) are the formulas used to calculate the return value.
The last function that changes a shape from a sphere to a cube and then to a torus is:
mix(mix(c3,c2,cf),mix(c2,c1,cf),cf)
compare sine function
SDF of a sphere, c1.
SDF of a cube in c2, and
SDF of a torus in c3.

Lighting:
We employed "View-Dependent" illumination. The lighting will adjust to the scene as the camera moves through it. This method produces dynamic lighting effects, but it can also present certain difficulties, such improper shadowing.

Results

Shadertoy Implemnetation link:
https://www.shadertoy.com/view/DlyGRh

YouTube video Link:
https://youtu.be/rgtKqadaXws

How to Run:

Click "Play Button" under the IDE on the right side of the screen when the Shader Toy Link is open.

On the left side of the screen, you can see the code's output.

Screenshots of the demo scene:

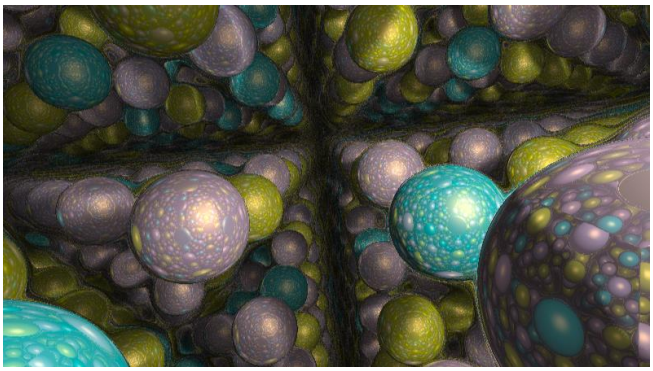

Figure 2: Sphere

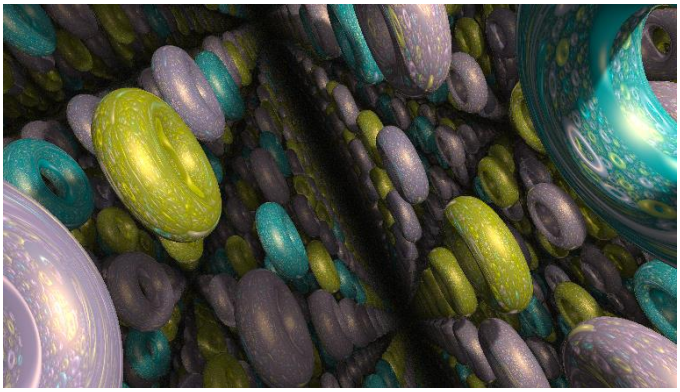

Figure 3: Torus

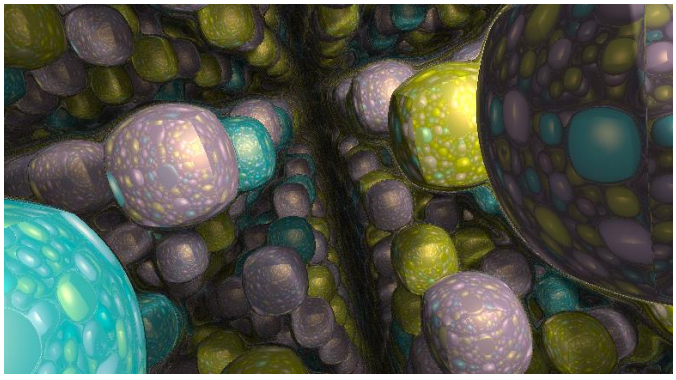

Figure 4: Spherical Cube

Conclusion:

After implementing the demo scene, we gained an understanding of how pixel shaders function. We studied the Ray Marching, Deriving Signed Distance Functions, Texturing with SDFs.