Signed Distance Functions

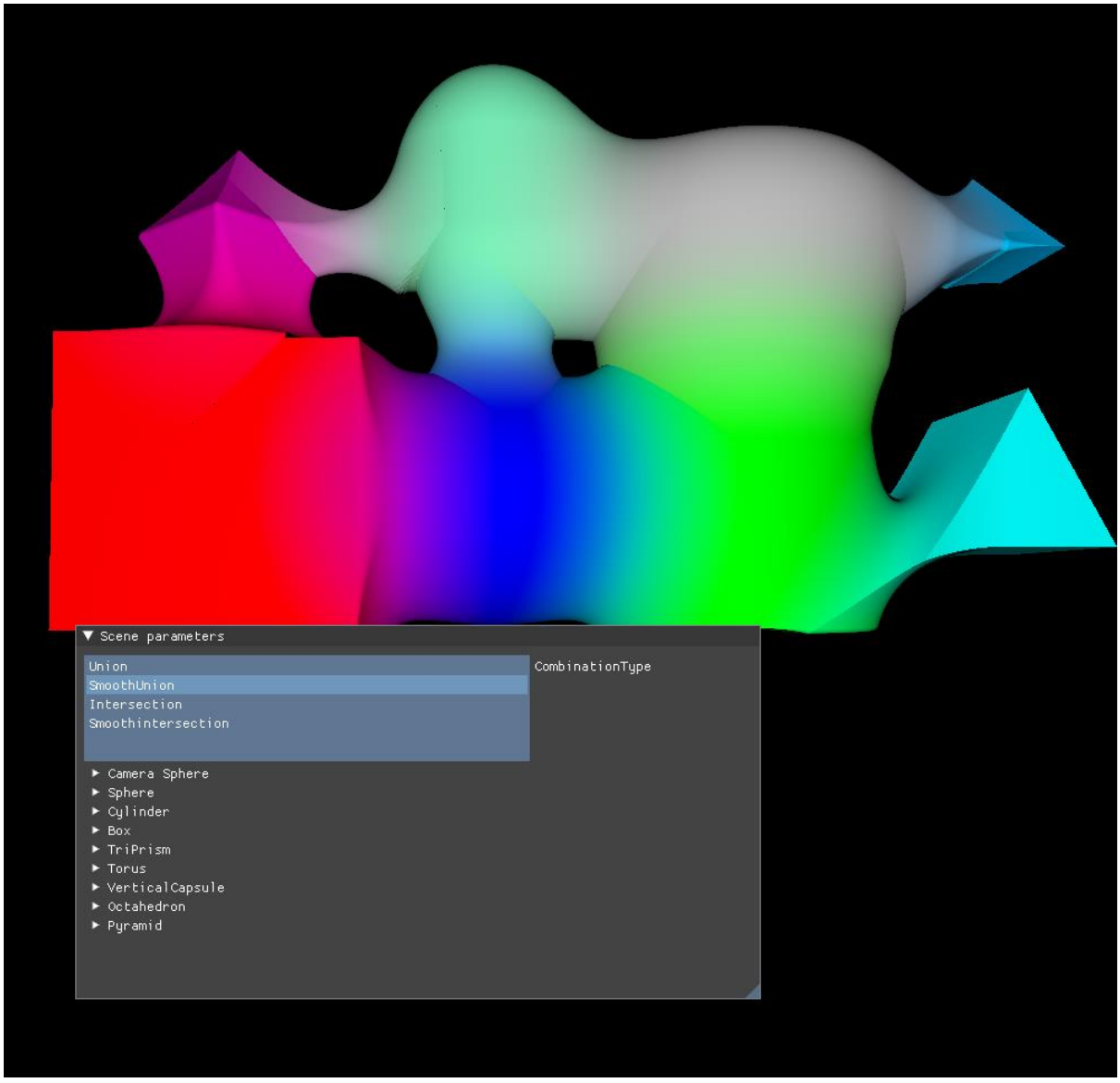
Andreas Pries-Brokmann - prie@itu.dk

IT University of Copenhagen

Course: Graphics Programming (Spring 2024)

Course Code: KGGRPRG1KU

15/5/24



Motivation

I have always enjoyed toying with visualization and how changes of parameters affect what is shown. SDF's having easily modifiable parameters and many possible shapes and interactions make them a great choice to create a program where you can play around with simple shapes and how they look/react when their parameters change.

Goal

Build upon the code provided in exercise 10 to expand the program with more shapes and options. This includes restructuring the code to add new shapes more easily, to have full control over the shapes, change what type of combination is used, if smoothing is used and to what degree.

Settings

This project is centered around the settings you can change during runtime. There is a GUI element called "Scene parameters" where you can view these options, they are split into CombinationType and one for each of the specific SDF's.

CombinationType controls which type of combination is used. The options are union and intersection as well as their respective smooth options.

There are 8 SDF's placed in the "world" and 1 that follows the camera. While playing around with adding new shapes I accidentally made one of my new ones follow the camera and thought it was quite fun, so I decided to add it to the final version. The camera specific one has the same options as the rest.

1. Smoothness: Sets the smoothness parameter for smoothunion and smoothintersection
2. Blend: Toggles whether the colors should blend
3. Enabled: Enables or disables the SDF
4. Matrix/center: Describes where the shape is located
5. Rotation: The rotation of the shape (where applicable)
6. Radius/Height/Width/Main radius/Side radius: Parameters to change depending on the SDF.
7. Color: The color of the object
8. Pyramid specific, broken base: A specific toggle to not apply the fix for the pyramids base looking like broken glass (a fun thing I found while implementing it that I decided to leave in)

To collapse or expand a specific SDF's options you can double click the name

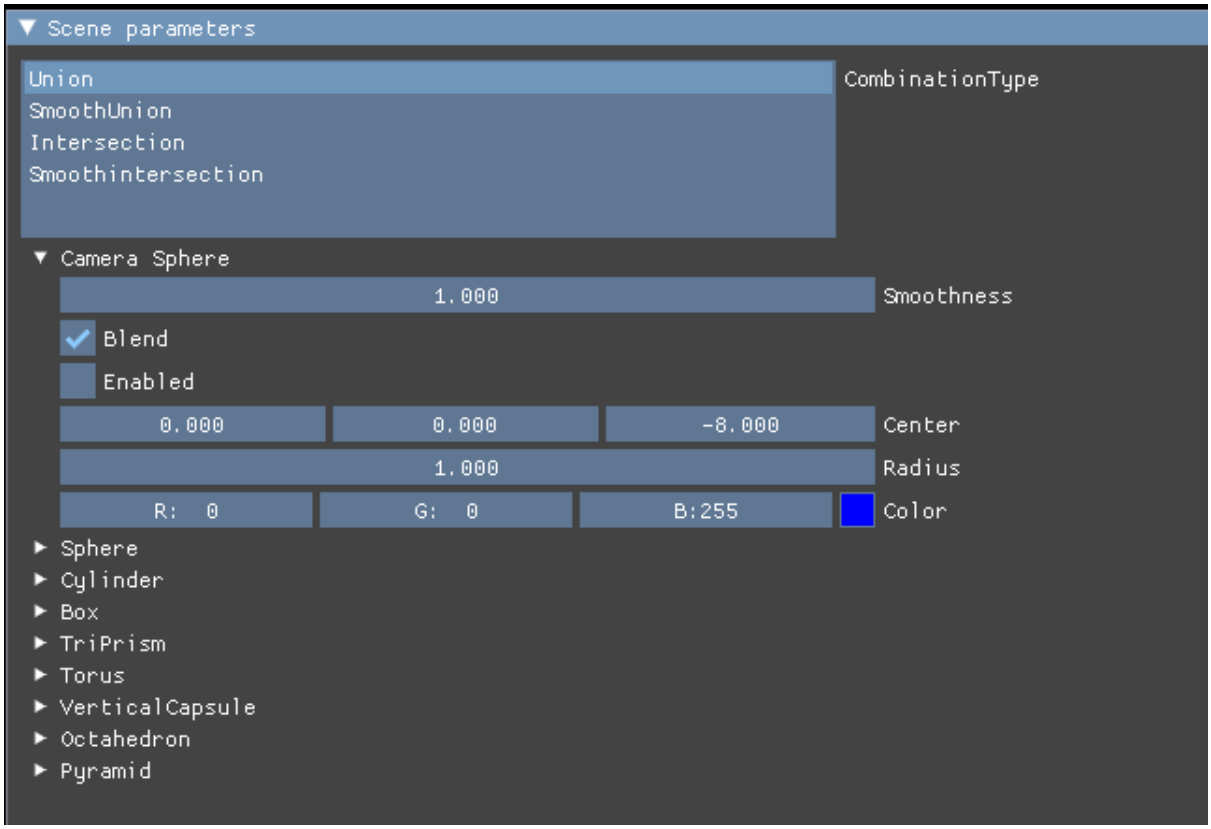


Figure 1: Scene parameters

Theory

Signed Distance Functions (SDF) will tell you the shortest distance between a point and some surface. This is used to determine whether a point is inside (negative number), on the surface (0) or outside (positive number). Combine this with raymarching, where you sample points along a ray (from the camera in this project). This point sampling (march) can be done with variable distance and finishing once the step is too small. This variable distance will be the minimum distance to any object in the scene, to figure out the distance that can be advanced without colliding with any object. This specific technique is called distance-aided ray-marching. *(De La Cruz, Gabriel. "Ray marching and SDF")*

Implementation

My implementation builds upon the work done in exercise 10, which means the overall structure and files used will be the same. Additional SDF shapes and combinations can be found in the "sdflibrary-additions.glsl" file in the shaders folder. The rest of the shader changes can be found in the "sdf-main.glsl" file. I have furthermore changed RaymarchingApplication header and cpp file.

sdflibrary-additions.glsl

I have added 4 shapes and 1 combination from iquilezles.org (Inigo Quilez. *distance functions).*

The SDF's have not been altered besides the pyramid function which has an optional condition to enable the broken base. I went with some new distinct shapes to be able to see how they blend with smoothunion and to have some variance in how they look. I tried different combination types (like subtraction) but I could not get them to work and decided to focus my efforts elsewhere. The math in these functions is complicated so I decided to use functions already created by others.

Sdf-main.glsl

I start by declaring all the uniforms used, the values assigned will be overwritten in RaymarchingApplication.cpp. I still use the output structure but decided to add my own SDFHelper to contain the different values used by a specific SDF to more easily add new ones, and to facilitate future improvements that will make the program more dynamic. I calculate the distance for each of the shapes, create a SDFHelper instance for each of them and add them to a list to be iterated over to find the 2 closest shapes to be used in the combination. This is not the smartest (or prettiest) way of finding the 2 smallest values of a list, but it works. If any object is disabled it will not be considered, but if less than 2 are still enabled it will default to just showing the 2 defined as closest and secondClosest. The blend value is for the color blend (if picked), I left this an option mostly for comparison sakes, since it looks quite different. The smoothness used in the combination is the average smoothness of the 2 shapes, this allows you to have different smoothness depending on which shapes combine, allowing some freedom.

My main problem with my implementation is how rigid it is. Although it improves on the flexibility provided in exercise 10, it still does not allow for the c++ code to add new shapes without modifications in the shader, by both adding the uniforms, the SDFHelper instance, appending it to the list, increasing the list size and the loop iteration size. But this implementation is fairly simply and served my purpose well where the number of shapes is limited.

RaymarchingApplication

First, I moved the conversion of the transformations to a separate method called SetViewMatrixRelation so that it is separate from the renderGUI step. This was mostly done as a separation of concerns. After that I added a lot of fields (anything that is not a float) as fields for the class. The camera sphere (and triprismHeight) does not need to be in world space and therefore don't have the viewmatrix multiplication.

I have then added all these values and the uniform values required for sdf-main in the initializeMaterial method, any value defined as a field in the header will overwrite the value defined here.

The rest of the additions are made in the RenderGUI method where I have added a new group for each SDF and the combination type selection at the top. This was done following the same structure I was given in the exercise, ideally this would be handled differently in the future, which I will talk about in the future improvements section. The way bool's are handled is ugly and is a result of how Booleans are treated differently and thus need to be converted, this could have been a simple function to not duplicate as much code.

Future improvements

The first two additions I want to make to my program is making it more dynamic, allowing for no change to the glsl file when adding new shapes, which would require giving it an array of values instead of setting each uniform directly at the top and figuring out a way to iterate over and create a list of variable length (or fake it somehow). This would tie together with cleaning up the renderGUI method and initializeMaterial method, by making classes for each shape and having a list of those shapes in the main class, allowing for more clean additions and shape repetition, possibly leading to being able to add more shapes directly inside the program. This would also be a more extendable approach and easier to maintain for the future.

I would also have liked to add more combinations and more control along with more shapes. This is not impossible to do but I felt the addition of more shapes should only come after fixing the above suggestion, thus making that change easier. I could not get other combinations (such as subtraction) to work, but that would have made a fine addition to the controls. There are also fun things like the broken base from the pyramid that could have been fun to add for other shapes by playing with their functions more.

Bibliography

1. De La Cruz, Gabriel. “Ray marching and SDF” Learnit.itu.dk. 18/4/2024

2. Inigo Quilez. *distance functions*. Retrieved May 13, 2024, from https://iquilezles.org/articles/distfunctions/