SpaceTurtle

Features

Nombre Asignatura: Motor Gráfico
Módulo Asociado: Unit 46 (L5)
Profesor: Arnau Rosselló
Curso: 2022/2023
Autor/es: Pere Prim Carol

Índice/Index

1.- Engine features

1.1 Entity Component System (ECS)

This system is the Core of our engine, since it is the central pillar of the operation of the entire engine.

And this consists of having a manager class that manages all the components that are in play, which will then be passed to the systems, so that they can easily traverse them and perform their respective actions.
This makes the memory continuous and speeds up the traversal of those components.

And the most basic, we have the entity, which is the identifier of which components belong to it.

For its operation, we have 2 basic classes "GameObj_Manager" which is the controller, which manages the creation of objects and stores the list of components and all its access management, all tempered for greater flexibility, and then the "GameObj", which are the entities that with an identifier, we can know which components it has.

And on the other hand, we find the components, for example the following:

1.1.1 ST_Camera

It is the component that is applied to an entity that, as the name says, makes it have the values of a camera.

1.1.2 ST_Collier

It is the component that is applied to an entity to have a collision, it has the maximum and minimum values of the collision size.

1.1.3 ST_Hierarchy

It is the component that is applied to an entity so that it can have hierarchy (Parent and children). It comes by default in all created objects.

1.1.4 ST_Light

It is the component that is applied to an entity to have light. It has the possibility of being Point, Spot or Directional, and stores the information of the color, intensity and even its shadowmap.

1.1.5 ST_Name

It is a simple component to give the object a name, it is mostly used for the internal HUD. This component also comes by default in all created objects.

1.1.6 ST_Render

It is one of the most important components, since it has all the data to render on screen the object to which this component is associated.
This component has the information of the mesh, the color, its textures, among other data.

1.1.7 ST_Transform

Another of the most important components, since it gives you the power to position on screen, have a size and rotate the object to which this component is given.

And then, we find the systems, where, by passing the list of necessary components, they do the necessary actions to fulfill your requirement.

1.1.8 ST_System_Assets

This system is in charge of loading and storing all the textures and meshes that have been loaded in the project, so that they can be accessed from anywhere.

1.1.9 ST_System_Camera

This system takes over from the list of cameras provided by the game manager, runs through them all and updates them.

1.1.10 ST_System_Hud

This system is responsible for displaying various menus, either the hierarchy menu, the stats menu or the inspector menu. All this, with the information of the components provided by the game manager.

1.1.11 ST_System_Lights

This system is in charge of going through all the light components, and updating its information and saving it in a light vector, with its shadows, to facilitate the rendering system and pass it to the program.

1.1.12 ST_System_Picking

This system is in charge of checking through the mouse position, the transformation components and the collision component, if the mouse is over an object, always returning the one that is closer.

1.1.13 ST_System_Render

This system is where more things are done, since it is the one in charge of showing something on the screen.
To do this, it accesses various components and makes different calculations to know which objects are to be rendered, and how.

1.1.14 ST_System_Transform

This system is responsible for calculating all the transform components.

1.2 Obj Instancing

This feature is mainly a performance improvement, since its main functionality is to draw the same object at once, instead of drawing them 1 by 1, which implies a great cost of calls to graphics.

To do this, it goes through all the objects to be rendered and as long as they are equal to the previous one, it stores them in a list until it detects a different object (either by its mesh, or any of its textures), then it paints all the objects stored at once in the list and continues the process.

In this way, you can go from drawing about 10,000 objects, to up to 200,000 objects (low polygon).

All this is done internally by the engine, without the user having to do anything, either with basic meshes from the engine, or with meshes loaded from any ".obj".

1.3 Hierarchy Transformations

This feature consists of being able to have hierarchy with the transformations, which in a simple way, implies that the transformations of the parent object are applied to an object.

It is easy to use, by accessing the component "ST_Hierarchy" we can add or remove "children", as well as access them, or their corresponding "parent" in case it has one.

1.4 Load OBJ's

The facility to load meshes from ".obj" files by saving them in the "st_geometry" class, which is part of the "st_mesh" inheritance.

It consists of, from the file path, accessing its information and saving it in a list of vertices and indexes, and saving these buffers in the ID that represents the mesh.
This can be easily done by accessing the "loadFromFile" function inside the "st_geometry" class.

1.5 Lights (Directional, Point & Spot)

In the lights, we can find the 3 most important, the Directional, equivalent to the Sun light, the Point light and the Spot with which to illuminate our scene and objects.

To do this we only have to add the component "st_light" to any entity, and then configure its parameters.

1.6 ShadowMapping (Directional & Spot)

Now, with the generated lights, we will make the shadows, and for this we will create a depth texture from the light and then we will pass it to the shader to check if it is behind an object or not, and in this way, it will be shaded.

Right now, only the Directional and Spot shadows are available, having "cascade shadow mapping" with the Directional, which works in a way that generates 3 shadows, each of them covering more space, but at less resolution, and in the shader, accesses its respective shadow depending on the distance.
Using in this way, the shadow with more resolution when you are close, and the shadows with less resolution when you move away.

The use of shadows is only activated if once the light is created, the shadows are activated, previously it was automatic, but to reduce consumption, now it is necessary to activate it.

1.7 Load Textures

The facility to load textures from files, saving them in the "st_texture" class.

It consists of, from the path of the file, accessing its information and saving this buffers in the ID that represents the texture. It also saves its width, height and number of channels.

Cubemaps can also be loaded.

1.8 Texture Atlas

This feature is another of the features focused mainly on optimization, although obviously it is also very useful. And it consists of, from a texture (TileSet) indicate in how many rows and columns it is divided, and to the component "st_render", we indicate which index uses that texture.

In this way the objects would use the same texture, but the same piece of it will not be displayed.

This together with the instancing is a perfect ally to be able to instantiate thousands of objects with "different textures" at the same time.

Or even sprite animations can be made, modifying the texture index used at runtime.

1.9 Skybox/Cubemap

We can also load cube maps, with which we can make 360º backgrounds. To do this, we will create a normal "st_texture", but we load the path with another function called "loadCubemap", passing a list of the paths where to find the images that form the cubemap.

Then, to use it, we simply add the texture to the albedo of the "st_render" component, and we use a program that accepts cubemaps.

1.10 Material Configuration

This is simply the possibility, having selected an object, to be able to modify all its rendering characteristics while the project is running, thanks also to the HUD.

In this way, you can change the textures, the mesh... with the ones loaded in the "st_system_assets".

1.11 Translucent

The ability to render transparent objects is quite important, either to create "windows" or ways to see an object through another, such as making sprites, and thus not see the background of it.

This option is activated from the "st_render" component and there is the possibility that these are ordered to always show the transparencies correctly, although with a large number of translucent objects, this can create a significant lag.

1.12 Picking System

This system is simple, but its functionality is super useful, since it allows you to select an object within the scene by clicking on it, which allows you to debug in a very comfortable way, and together with the HUD where you can modify things of that object, we would have it all.

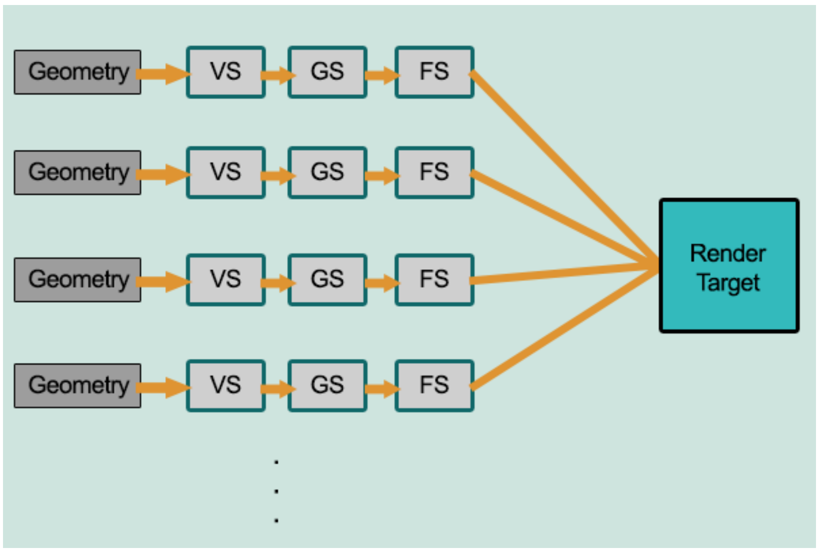
To use it, you just have to call the function when you want to check if you are clicking on an object, and it will return the ID of the object clicked, in case of not clicking any, it will return -1.

This system works thanks to the component "st_collider" and "st_transform" which will check a ray launched from the mouse position and check that line if it intersects with the collision box created by the component.

2.- Implemented Techniques

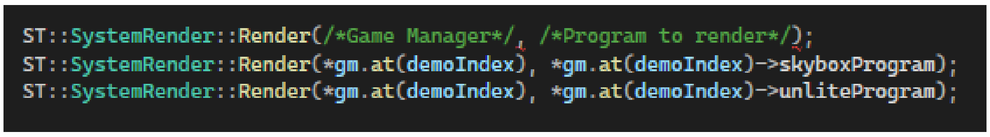
2.1 Forward+ Pipeline

The forward pipeline is the simplest rendering system. And its operation is that each object with a mesh and a program, are rendered directly on screen. Including if you want to illuminate, this will have to calculate each of the objects by itself, having a clear limitation of the array that you can pass to its program.



This apart from limiting the amount of lights per object, and even the shadows, also prevents us from being able to make more complex calculations such as postprocessing, among others.

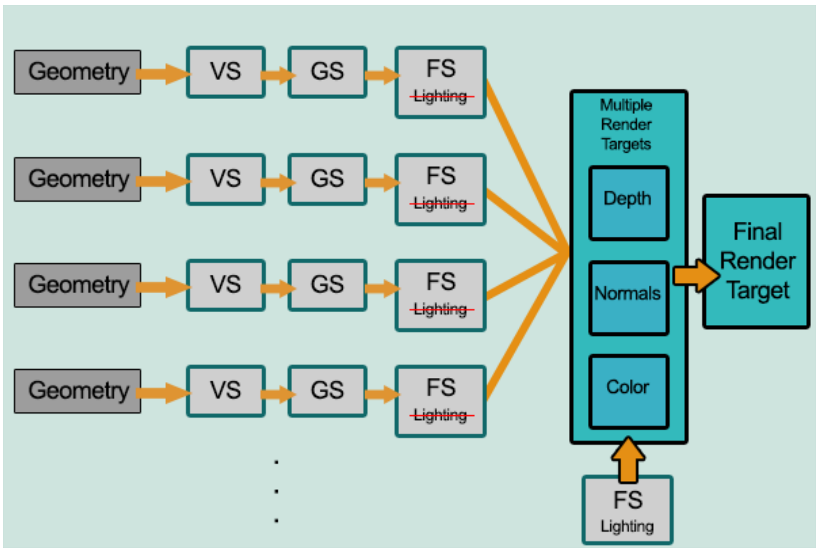
This is the basic rendering of our engine, and for this, you only have to call the "st_system_render", indicating which program you want to render, so it will go through all the objects and render those that have that program.



```
ST::SystemRender::Render(/*Game Manager*/, /*Program to render*/);
ST::SystemRender::Render(*gm.at(demoIndex), *gm.at(demoIndex)->skyboxProgram);
ST::SystemRender::Render(*gm.at(demoIndex), *gm.at(demoIndex)->unliteProgram);
```

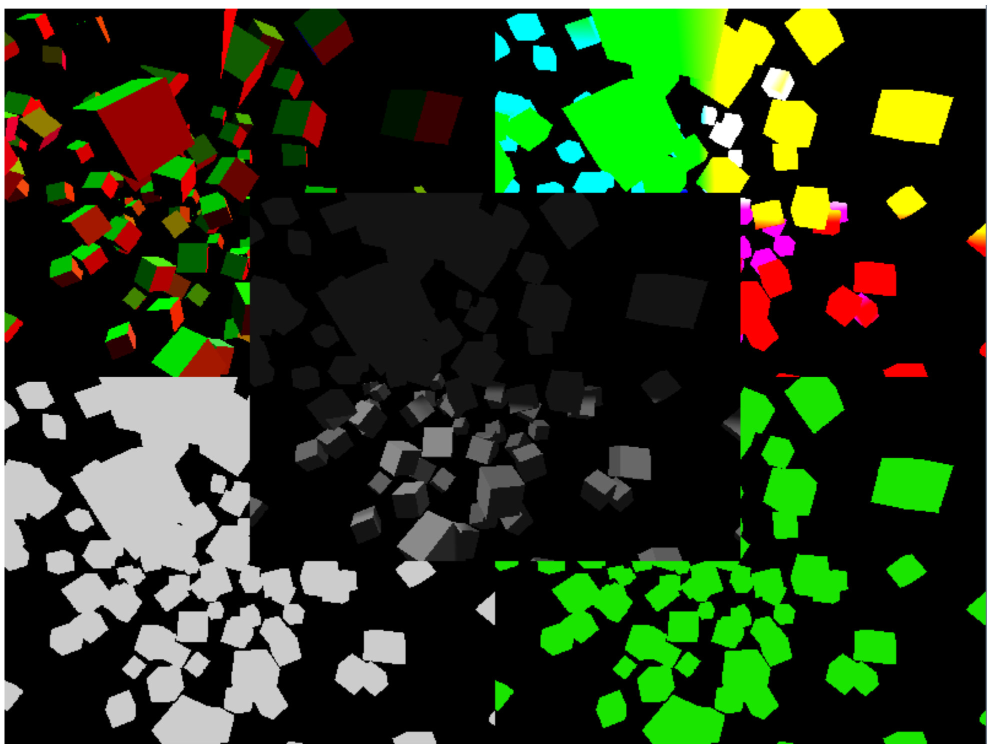
2.2 Deferred Pipeline

This pipeline is already more complex, but apart from being more efficient in some aspects, it gives the possibility to make more complex effects and post-processing.

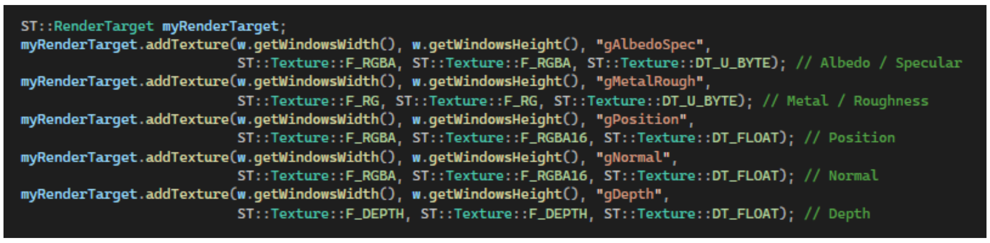


For this, its operation consists of making a first pass of all the objects in the scene, but instead of painting the information on the screen, saving in textures the basic information, such as: the position, the normals, the base color, among others...

And then, with all that information, it is passed to a final program, where it will be rendered in a simple quad on screen, making the calculations you want there, such as lights, shadows or even post-processing effects such as SSAO, blur, among others...

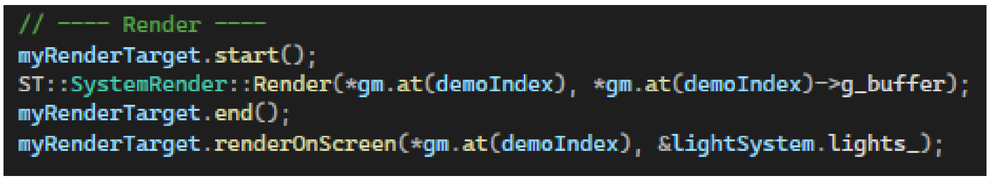


To make use of the deferred pipeline, by default the objects created with the "st_render" component, we introduce the program to make the deferred, which is the "g_buffer", and in this way we only have to prepare the "st_rendertarget" adding the textures that we want to create, in this case like this:



```
ST::RenderTarget myRenderTarget;
myRenderTarget.addTexture(w.getWindowsWidth(), w.getWindowsHeight(), "gAlbedoSpec",
                          ST::Texture::F_RGBA, ST::Texture::F_RGBA, ST::Texture::DT_U_BYTE); // Albedo / Specular
myRenderTarget.addTexture(w.getWindowsWidth(), w.getWindowsHeight(), "gMetalRough",
                          ST::Texture::F_RG, ST::Texture::F_RG, ST::Texture::DT_U_BYTE); // Metal / Roughness
myRenderTarget.addTexture(w.getWindowsWidth(), w.getWindowsHeight(), "gPosition",
                          ST::Texture::F_RGBA, ST::Texture::F_RGBA16, ST::Texture::DT_FLOAT); // Position
myRenderTarget.addTexture(w.getWindowsWidth(), w.getWindowsHeight(), "gNormal",
                          ST::Texture::F_RGBA, ST::Texture::F_RGBA16, ST::Texture::DT_FLOAT); // Normal
myRenderTarget.addTexture(w.getWindowsWidth(), w.getWindowsHeight(), "gDepth",
                          ST::Texture::F_DEPTH, ST::Texture::F_DEPTH, ST::Texture::DT_FLOAT); // Depth
```

And then, to render, we will make use of the forward pipeline, but making that information be saved in the render target like this:



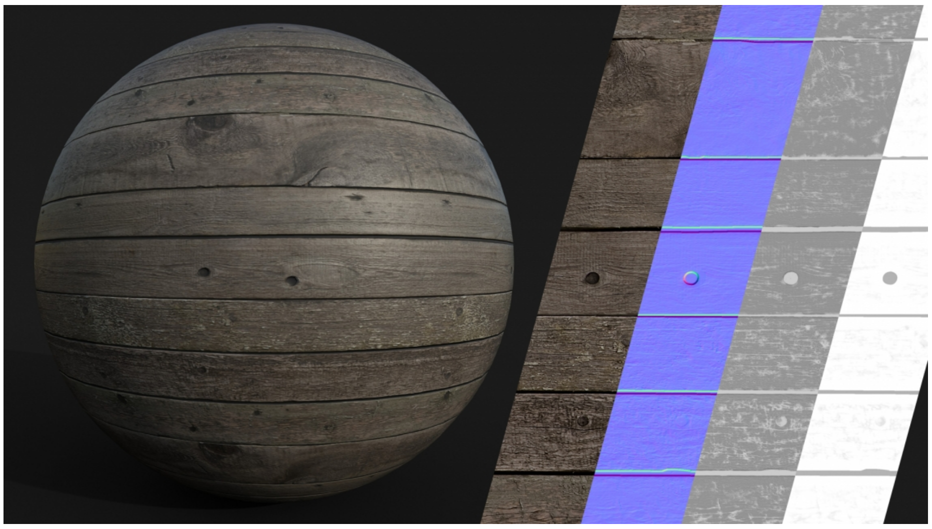
```
// ---- Render ----
myRenderTarget.start();
ST::SystemRender::Render(*gm.at(demoIndex), *gm.at(demoIndex)->g_buffer);
myRenderTarget.end();
myRenderTarget.renderOnScreen(*gm.at(demoIndex), &lightSystem.lights_);
```

And within the "Render On Screen", is where the light pass will be done, and even, in this case, the SSAO pass, but any other type of post-processing could also be done.

2.3 Materials PBR

PBR materials is simply calculating the lights from physical calculations, with the use of more textures and object information, such as the Roughness texture, which indicates how rough or smooth the object is, and the Metallic texture, which indicates how metallic it is.

With that, and some new light calculations, a much more realistic visual effect is achieved.

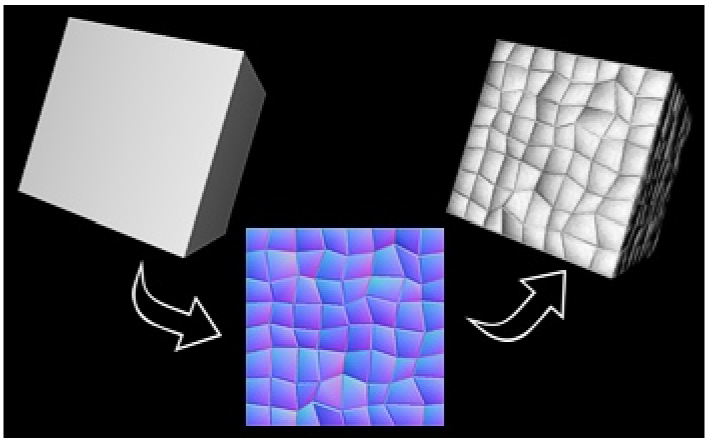


2.4 Normal Mapping

With normal mapping what we can do is to simulate that the object, even being "flat" seems to have relief, and the light affects it in such a way.

This technique, mainly serves to optimize, since its function is to reduce the polygons of a mesh, but without losing the detail of how the light would affect it in those small details that we might want.

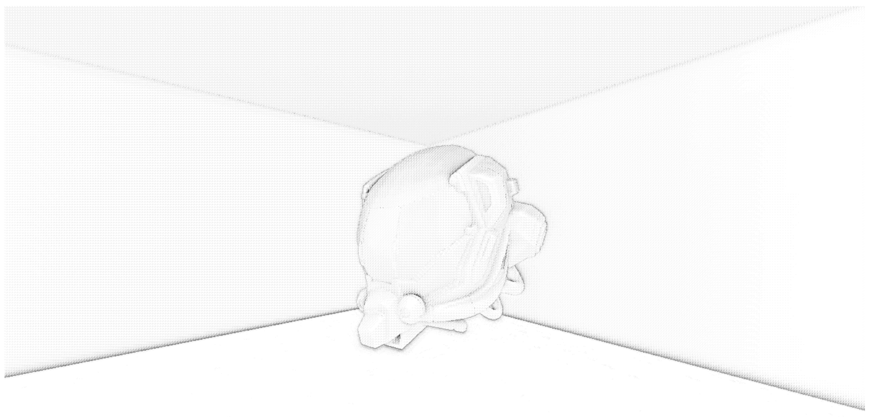
To do this, we will do it with a texture taken from its high Poly version, and internally then that texture is interpreted as its normals with which the light reacts as if it really had that relief.



In our engine, this is handled automatically, depending on whether the object in its "st_render" component has the normal texture or not.

2.5 SSAO (Ambient Occlusion)

This technique is a post-processing, which simulates an approximation of an indirect light, and what it does is to darken the parts that are close to each other.
This effect is sometimes not very obvious, but it is very noticeable when activated and deactivated.



Its operation within the engine is also automatic, although it can be activated and deactivated from the "st_renderTarget".

3.- Strategies and Solutions

3.1 Strategies

To carry out this project, we have not followed a very solid strategy to do it alone, but we have always tried to follow a pattern of going for a clear objective, and until we finish it, not to go for another one.

Even so, that does not always end up being this way, since there is always something that gets a little hung up, you go ahead with another functionality, and later, you find how to solve this one.

Also, the positive part of doing it alone, is that I have been able to deal with things at the pace that was best for me, and to focus on the feature I wanted. Although, I would also have liked to have more support from my colleagues and to be able to share knowledge.

3.2 Solutions

During the production of the project there have been several minor problems, which have been solved without much trouble.

Even so, some to highlight that have given more problems have been:

Problem with the rotations of the transformations, this was dragging until I managed to realize and solve it, and from here, other problems were solved.

Another more recent problem has been when implementing the SSAO with the Deferred pipeline and the additive light, which, following the tutorial of "LearnOpenGL" did not work, and I had to do it in a very different way, completely changing the shader.

And finally, I would have liked to implement the Skybox reflection to the PBR materials, but it didn't give the expected effect, and finally, I had to abandon that part for the moment.

4.- Tasks Distribution

Pere Prim Carol

Bibliografía

Webs

- https://gamedevelopment.tutsplus.com/articles/forward-rendering-vs-deferred-rendering--gamedev-12342
- https://www.3dgep.com/forward-plus/
- https://google.github.io/filament/Filament.md.html
- http://jultika.oulu.fi/files/nbnfioulu-201805101776.pdf

Images

- **00** https://i.stack.imgur.com/8WwEh.png
- **00** https://uploads.jovemnerd.com.br/wp-content/uploads/2023/01/this_is_fine_capa__wg1a6r.jpg
- https://www.goodtextures.com/cache/6d37659b/av8c5cfec6f0ed2bd9f65.jpeg
- http://www.independentdeveloper.com/images/normalmapper_thumb.jpg