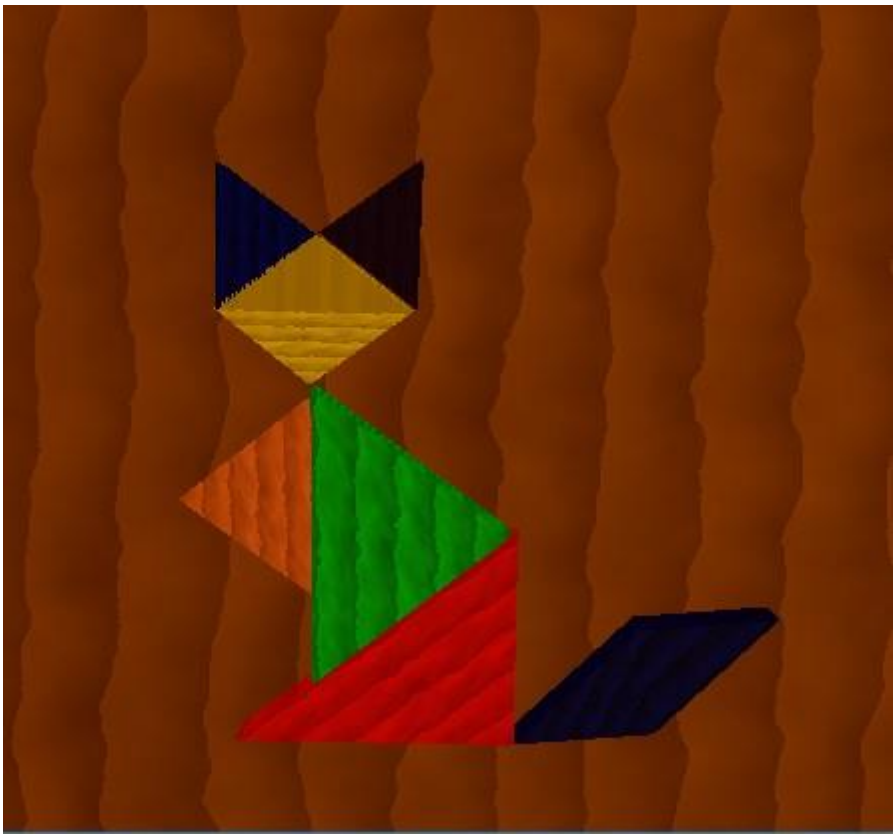
Computer Graphics for Games 2018

Wood Tangram

A project where it is possible to create your own tangram figure in real time.
Where the table and the tangram pieces are made of realistic wood.



João Henriques
81633
CGJ
j.tiago.henriques@tecnico.ulisboa.pt

Mário Santos
79769
CGJ
mario_jps@hotmail.com

Abstract

Since in the individual project we already had an application that with the press of a key passed from a closed tangram to a figure, it made perfect sense to create something that would allow this to be done in real time. So, we decided to make a project that allow that. And to make the scene more pleasant to watch we chose to make the scene as realistic as possible using textures, normal mapping and Perlin noise generated wood for the pieces and the table.

For every problem we had, we always chose the solution based on his simplicity while also trying to maximize the realistic look of the scene.

During this report we will show you our challenges, the problems we encountered and our solutions to these same problems.

1. Concept

Our concept in a resumed way is a continuation of the individual project. Is a project where you can play with a virtual tangram. You start with a cube and you can make all the possible shapes that tangram allows.

Figure 1- initial tangram

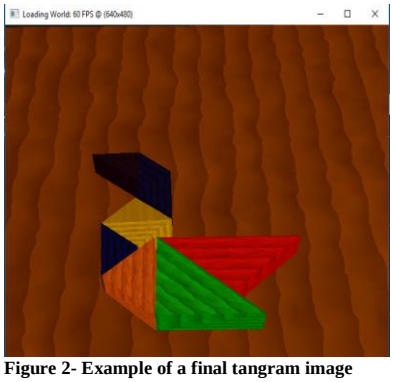


Figure 2- Example of a final tangram image

We tried to make an interactive, easy to use and at the same time realistic looking We also give a couple of features to make the experience more enjoyable, such as snapshot your work and load/save your scene. With that you can share your work with others and continuous in another day. It's also possible restart your scene, and if you are perfectionist we give an option to approximate the angle of each piece to the units. For the realistic look we used Blinn-Phong model. To make our virtual tangram as similar as possible to the real tangram, we apply a realistic wood, to all the pieces and to the table.

2. Technical Challenges

2.1. João Henriques Challenges

2.1.1. Generic scene graph handling hierarchical drawing

2.1.2. Use Stencil buffer to pick objects

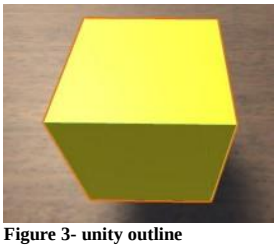


Figure 3- unity outline

2.1.3. Allow to save and load the scene

2.2. Mario Santos Challenges

2.2.1. A "photorealistic" lighting / shading model, Blinn-Phong model

2.2.2. Shader based special effects, normal mapping

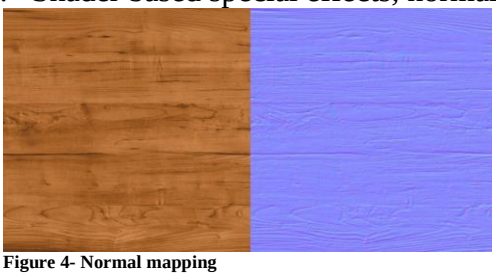


Figure 4- Normal mapping

2.2.3. Saving a snapshot of the application to a known image file format

2.2.4. A realistic or stylised solid material for the objects of your scene, Wood

Figure 5- realistic wood

3. Proposed Solutions

3.1. Generic scene graph handling hierarchical drawing

3.1.1. I used that point to remake everything that was bad from the individual final delivery. We start to see what ass necessary to fix for our project, and what as the importance, so we could set the priorities.

3.1.2. There were several bad things in our initial engine. We didn't had the initial tangram done, the parallelogram.obj and the triangle.obj were not correctly centred in origin. We used quaternions and Rodriquez formula to rotate the objects. Our model matrix it was well calculated, but the code was ugly we wanted a more syntactic sugar code. All that problems where fixed, making it easy the rest of the program.

3.2. Use Stencil buffer to pick objects

3.2.1. We decided to use stencil buffer because it makes possible to select the piece and make a outline in the piece selected. Also have the great advantaged that It's easy to select a piece even when you have two pieces together, which is almost all the time in this project

3.2.2. Our project is basically composed of 9 pieces (7 tangram pieces, 1 table and 1 "lamp"). So, what we did was assign a value(index) to each piece, and when we were draw that piece we put that index in the stencil buffer. We used the function "glReadPixels" do read the index of the pixel selected, so we are able to know each piece we click on. Since the stencil buffer have every pixel with a index it makes easy to distinguish pieces that are together.

Since we already had the stencil buffer implemented to select each piece we used to make a outline when the piece is selected. Basically what we did was after the draw of all pieces, we draw the piece selected again, but that time we draw in black, and a little bit bigger, and we don't draw in the part that have the index of the piece selected, and this is how you can make a outline.

The only problem of stencil buffer is that only have 265 positions, but in our case is not important because we only have 9 pieces, so we only need 9 positions. There is no risk of filling the buffer or re-write so position.

In the tangram there is a special piece, the parallelogram. The parallelogram is unique because is the only piece that if you rotate only be allowed to rotate 360º in yy axis you can´t get all the possible draws. You need to be able to rotate 180º in zz axis to get the rest of the possible designs. So I need to take than in account when I'm selecting the piece.

Figure 6- Red triangle selected

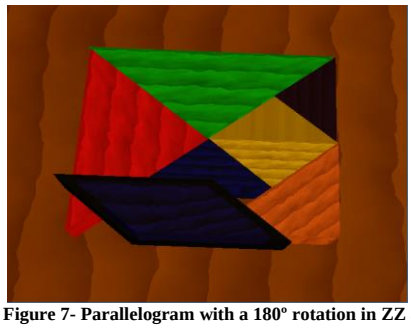


Figure 7- Parallelogram with a 180º rotation in ZZ axis

3.3. Allow to save and load the scene

3.3.1. To load and save there are diferente options (XML, database, .txt, JSON). A database is it is usually necessary to make a big setup and more computer processing. A file .txt does not give any advantage of the fact that no library is needed. From XML to JSON there is not much diference, and since I already work alot with JSON (not in C++), I decide to use JSON.

3.3.2. There are alot librarys to work JSON in C++, i decided to use jsoncons (A C++, header-only library for constructing JSON). I choosed that beacuse it was not necesary any setup and had a few exemples in her github page.

At the beginning was hard to use JSON in C++, especially because in C++ you need to know the types of variables that you are reading and because there was no function to directly read arrays from the JSON file. Also the code was a litle bit static, so it was necessary to create some code, especially one shaderManager and one meshManager that would make possible to find the shader and the mesh by name.

After that initial strungle it was easy to do the rest of the challenge.

In a more detailed way, i use two json files, "defaultLoad" and "savedScene". If the last file doesn't exist, means the user never saved a scene so automatically load the default scene. If the two files exist the program ask to the user what file should be loaded

I also decided to implemented a restart option, that restarts the scene to the scene loaded.

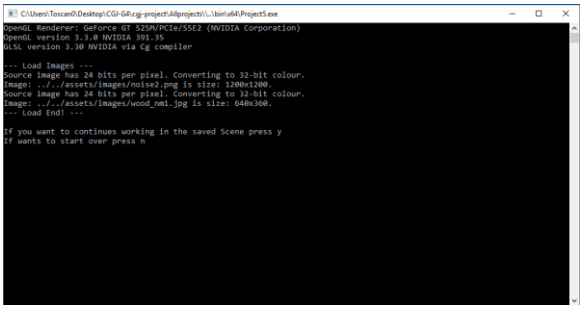


Figure 8- Ask to the user what scene should be loaded

3.4. A "photorealistic" lighting / shading model, Blinn-Phong model

3.4.1. We decided to use Blinn-Phong model for realistic lighting, the alternative was the Phong model but we thought that Blinn-Phong would look more realistic since it gives an elliptical specular light instead of a circular one.

I implemented the model the regular way, I created a light point and transferred it as a uniform to the vertex shader. I had to change some parts of the code, namely the camera class is the shader class, to be able to do this as the code as quite static.

In the vertex shader I create the camera view being the position in view space, then in the fragment shader I set the Ambient, Diffuse and Specular, light intensities and material reflectance. And proceed with the calculations of each component, for the ambient it's just the material reflectance times the light intensity, for the diffuse it's "`clamp(dot(N, L), 0, 1)`", being N and L the normal and the light vectors, respectively, times the material reflectance times the light intensity. For the specular component I create the half vector (H) which is the normalized sum of light and camera vectors, then it's "`pow(max(dot(N, H), 0.0), matShininess)`" times the material reflectance times the light intensity.

Finally, I just calculate the color with the contribution of all the components.

3.5. Shader based special effects, normal mapping

3.5.1. For this challenge I had to integrate a library for manipulating images, I end up choosing the FreeImage library, since I tried the Cimg library but it was too simple for what I desired. Then I created a class Image with a static method that could load images to the program.

I looked for images to make the texture of the wood, I chose one and through GIMP and a plug-in, I created a normal map through that texture. Then I proceeded to load the texture and the normal map to the program and pass as a uniform to the fragment shader when I extract the diffuse color from the texture and the normals from the normal map and convert the normals to values between 0 and 1.

In order to make this work I had to create a function in the mesh class to compute the tangent and bitangent of each vertice and then store them in buffers. The way I do this is through this formula:

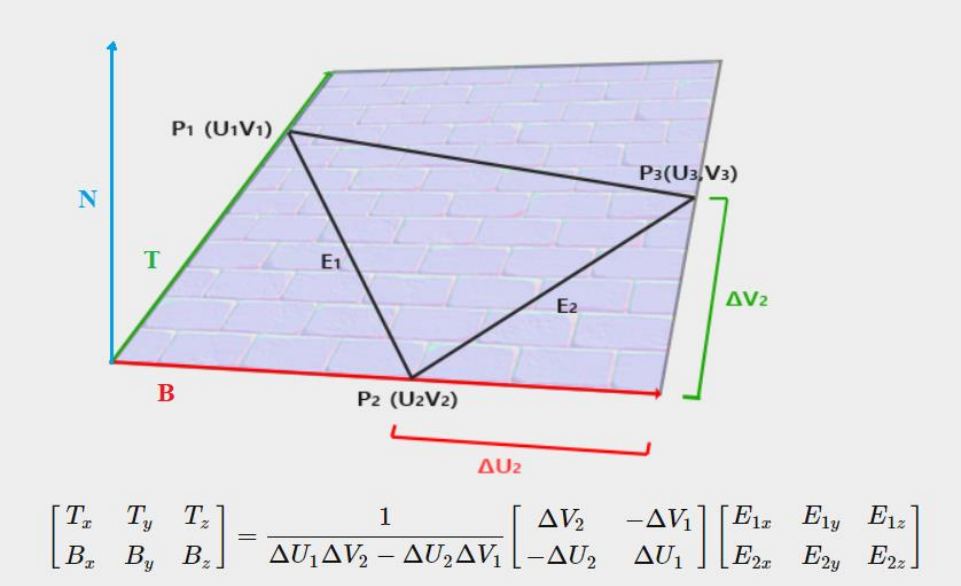


Figure 9- Formula for calculating tangent and bitangent

Now that I receive the tangents and bitangents in the vertex shader, I create a 3x3 matrix, called TBN with the tangent, bitangent and the normal(the normal from the mesh, not the one from the normal map) which will allow me to convert positions to tangent space, so a use it to transpose the light and camera position from view space to tangent space.

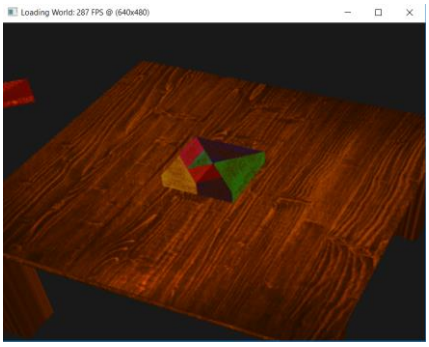


Figure 10- Wood texture applied

Figure 11- Specular light reflecting with normal mapping

3.6. Saving a snapshot of the application to a known image file format

3.6.1. In order to create a snapshot I added to the image class a snapshot counter and a method to snapshot. This method uses the function `glReadPixels` to get the color of the pixels, then I use the method `FreeImage_ConvertFromRawBits` to convert the bytes of the color to FreeImage format and finally I save it with `FreeImage_Save` method as an JPEG image with the name *"Snapshot#"* in the image folder in the assets, the *"#"* being the order of the snapshot. To finish I associated this method with the *"P"* key.

3.7. A realistic or stylised solid material for the objects of your scene, Wood

3.7.1. We decided to use a method to generate wood that looked realistic instead of a texture image, so we used Perlin noise to generate a somehow naturally random wood.

3.7.2. First, I chose a Perlin noise image then I loaded it and passed it to the fragment shader as a uniform. In the fragment shader I get a noise value from the noise image and use this method to transform the colors: “`sin(fract(scale * exTexcoord.x + noise * linesNoise)) + noise * gradient`”, being the “`exTexcood.x`” the x coordinate of the texture position. And with this method I apply lines on the texture, being able to change the number of the lines changing the valuable `"scale"` changing the randomness of the color gradient (from dark brown to light brown) with the “`gradient`” variable and being able to change the curves of the lines with the variable `"linesNoise"`.
With this change I couldn’t use normal mapping because as the texture was being generated in real time I couldn’t generate a normal map or at least I didn’t find how to do it, which made the lightning mode somewhat useless.

3.7.3. I tried a couple of thigs to make the wood look more realistic because since the table was flat the method didn’t look realistic, I tried to use rotation matrix to transform the texture coordinates in order to change the look and create some ellipses in the wood but it was too complex for me to understand and I didn't get the visuals I intended.
In the end I sticked with the simple version as it was simple for me to understand even if it didn’t looked realistic.

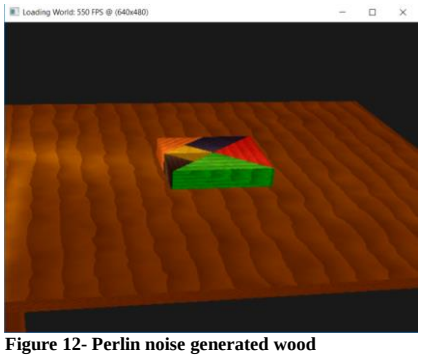


Figure 12- Perlin noise generated wood

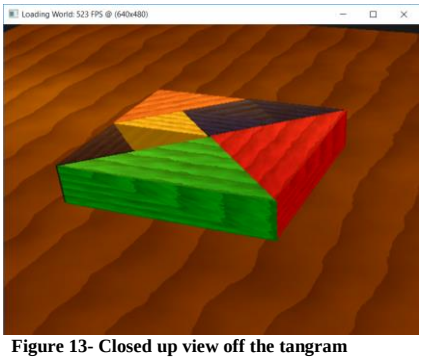


Figure 13- Closed up view off the tangram

4. Post-Mortem

In general, it was a hard project. It was in C++, a language that we are not used to work with. Me and my colleague are from a different campus and have different schedule making it really hard to meet. It had several technical challenges, that were very time consuming and a lot of effort was put into it. But at the same time, we learned allot and it's amazing to look back and see what we've done. An engine 😊

4.1. What went well?

The load and save scene were easy to make and gave me the desire to implement more related things, such as a restart and *“ctrl+z”*. The restart was fully implemented, but I did not have the time to implement the *“ctrl+z”*.

The snapshot and the normal mapping were technical challenges that in the end worked very well and did not gave much problems. The normal mapping was a lot

of fun to do and was the one that surprised me the most giving the scene another level of realism.

4.2. What did not go so well?

From my challenges (2.1.1, 2.2.2, 2.2.3) the only thing that was not so good was the outline. Is not as beautiful as I thought it would be, it was too big. Sometimes is hard to put a piece in the right position because the black line gives a wrong idea of the size/position of the piece. I guess is not hard to fix that problem, maybe just change the parameters of the outline design. Unfortunately, I didn't find the right parameters, maybe if I had more time, I could fix that.

The Blinn-Phong model even being supposed to be simple and easy to implement since there are many examples already made, it gave immense problems and headaches, and I still think that its implementation is somehow wrong.

Wood generation was quite a challenge too since most of the time this challenge took me, I was not actually coding but just trying to understand which function did what to the texture. And in the end, I sticked with the simple version, which I'm not very happy with.

4.3. Lessons learned

If I would go back, for the moment that I learned quaternions I would immediately change every rotation to quaternions. Because in the middle of the project there were some objects using rotations with quaternions and some using the Rodriguez formula, and some using both. That only complicated the code and added unnecessary bugs.

Preparing the code previously to be more flexible would have given us more time and spared future work so that we could focus on the technical challenges.

Some challenges required a great deal of theoretical knowledge, so it would have been wise to study the theoretical part more before coding.

References

For the stencil buffer outline I used that tutorial to learn:
https://learnopengl.com/Advanced-OpenGL/Stencil-testing

GitHub of jsoncons: https://github.com/danielaparker/jsoncons

For the image loading I've based myself in this solution:
https://r3dux.org/2014/10/how-to-load-an-opengl-texture-using-the-freeimage-library-or-freeimageplus-technically/

For the Blinn-Phong model I've based myself in this solution:
http://www.sunandblackcat.com/tipFullView.php?l=eng&topicid=30

In order to understand normal mapping I used this page:
http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-13-normal-mapping/#vertex-shader

For the wood generation I used this page to understand the method:
https://thebookofshaders.com/11/