Generating 3D solar systems by using pseudorandom number generators and OpenGL

R. Cuadra, L.C. Arias

Abstract Instructional media can help people understand the real life objects with the help of a different an unique perspective, which can be achieved with the help of computer graphics, where we can develop real objects inside unreal worlds and make them interact, in order to explain physics and concepts of the real life, we developed a 3D sandbox space simulation game where one can develop their imagination with a randomly generated universe that uses the same concepts of our universe when talking about solar systems, planets and spacecraft movements, we planned a whole immersive experience where the user can explore different auto generated regions of a galaxy using a spaceship and observe different possible systems with different materials and speeds, it is planned to be available on all the systems that support at least openGL 2. The whole project is attempting to be introduced into the either entertainment or learning areas. Product development still needs to be done primarily on the addition of the completeness of the material so it can be used also by the general user.

***Index Terms*—space , simulation, computer graphics, randomization, sandbox, 3D, open world**

I. Introduction

We strongly believe that the use of computer graphics in the education has been a real game changer and had helped lots of current products to achieve the production status, we implemented a space simulator where people can spend their times to learn the basics of physics and to let their imagination flow into an open and explorable world.

II. Procedures

A. Design Stage

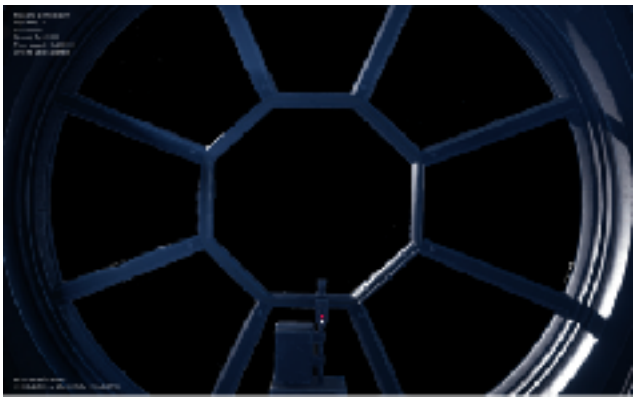
We started by doing some sketches of a common solar systems and what attributes make a system a system, after that the features were chosen, the chosen schema was Universe contains Solar Systems and each system contains Planets, where each planet has Moons and/or Rings. As well, we decided that inside the space there should be some spaceships wondering out there.

B. Work breakdown

The whole scene was planned to be rendered using OpenGL and to implement materials, lights, line strips, primitive figures, textures, transparencies, blendings, ".obj" models, shadows, movements over time, interruptions, audio and a lot of random numbers, which all together means even more calculations.

The biggest troubles we had was implementing an algorithm for reading the "obj" models and loading custom textures with transparencies, which we chose to integrate in a ".png" file, for the first part me decided to implement de "GLM: an Alias Wavefront OBJ Library" from Nate Robbins which has a really good and efficient algorithm for loading object files with an attached material and even some textures.

The second textures loading was solved using the "libpng" by Schalnat, Dilger and Bowler, here we obtained an algorithm that allowed us to generate a texture with an alpha channel from a png file, which was used for displaying a Head-Up Display



Another important part was developing the planets movement, which is the core of the simulation, there were 2 main movements in all objects inside de system, the

rotation and translation, one solved with the simple glRotate over time and the other required an equation that calculates the position in a circumference by giving the time.

Next to implement was the wandering spaceships, which involved 3 main problems, which were :

- Movement
- Pointing At
- Spacecraft Body (Object file attached to it)

And the first two were solved by some 3d vector calculations and the last one with the help of te previously mentioned library.

The last part was the audio and we used SDL2 for it to work, there is a simple class in charge of handling all related to music and it is pretty straightforward, as well the other points were easily implemented by using native OpenGL functions such as blending, illumination/reflection and windows/sizes/system interruptions.

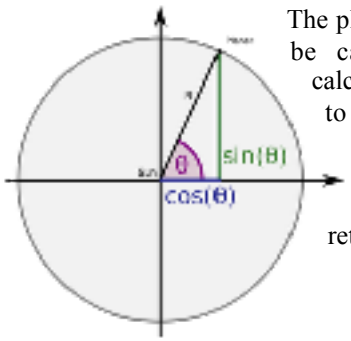
We have to add for the record that the camera and perspective were, at the beginning , meant to cause troubles, but we find out a repository on GitHub that gave us a really interesting camera with some already implemented controls that simulated an object in the space, so we adapted it without much trouble, meaning that the camera and views wasn't much work

C.Implementations

The first iteration involved developing de system and its movement, we created a class called Solar System which is a collection of planets and it has an *x*, *y* and *z* component; the first "planet" will always be a sun and it has a light attached to its position, then it can have from 0 to *n* planets, after all, this class has a function of array with extended features.

Planet Movement

The next class that we need to explain is the planet, this object has an orbit time, rotation time, distance from the sun, its radius and contains an array of moons and rings. The planet translation position is calculated by using :



The planet has a function that must be called before rendering, it calculates its position according to the sun, it requires the distance from it (R), the current time (t) and the time it takes the planet to return to the origin (o)

$$\theta = \frac{t * \pi}{o}$$

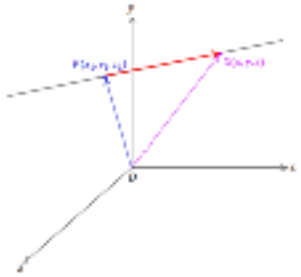
$$x = R\sin\theta \qquad y = R\cos\theta \qquad z = 0$$

it is important to know that all de distances are scaled from the real kilometers to a 0.0005% for aesthetics and efficiency, so we don't work with overflowing numbers. This method for calculating de translation is also used on the moons, where the angle is calculated according to the planet and the moon also has its own orbit time and distance from the planet.

The next movement is the rotation one, this one is a bit simpler, it is calculated with $rot = \frac{t * 360}{r}$ where r is the rotation time and it uses the same time. So now that this values were calculated the planet/moon/ring is called to be rendered, so the first thing is a translation to the calculated *x*, *y*, *z* from the translation step, then we do a rotation over the *z* axis and after that we draw the object.

Spaceship Movement

For this object we decided that every spaceship should have coordinates attached to it, the first one is a start point *P* the second one an end point *X* and the last one is the current position, the routes were decided as a simple straight line in a 3 dimensions plane, so we used the line equation



$$x_f = x_p + p\Delta X \qquad y_f = y_p + p\Delta Y \qquad z_f = z_p + p\Delta Z$$

The *p* is obtained from the time, it is the result from a modular operation that give us values between 0 and 2, the 0 to 1 mean that the ship is going forward (from P to X), while the remaining the opposite. With the calculated components we do a translation and thats all.

Spaceship Pointing

This one required a bit more, here we required 2 things, the ship vector, and the object to look (some coordinates).
the ship vector *S* is defined as <0,1,0> because it started as a cone looking to the sky, to rotate this to *F* <x,y,z> we implement $v = ||S \times F||$ and $c = S \cdot F$ And with this we obtain the rotation matrix $R = I + [v]_x + [v]_x^2 \frac{1-c}{s^2}$

After calculating the rotation matrix you just rotate it and the ship must point to the object

Spacecraft Body

The initial model for the spacecraft was a cone, but we modified that for it to be a real unidentified flying object, so we loaded an object file from some common known and free files and attached to it, is just a simple render.

III. RANDOMIZATION

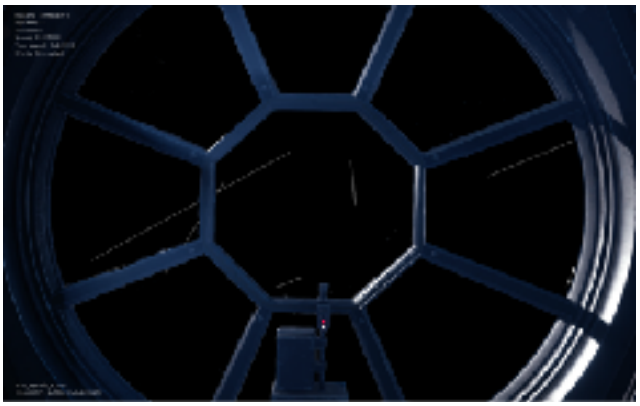
The main idea of the project was not only a space simulation but an infinite exploration sandbox, which could be possible by dynamically generating the scenarios, for this to be possible we replicated some features from a well known game named “No Mans Sky”, so we decided that for our code should use a pseudo random number generator, where the seed will be the identification of the created region of that galaxy. We implemented the default c random functions and developed an interface of the existing classes to wrap and work with the seed, so if we have a class named fleet(Collection of spaceships) then we have a random fleet(Collection of random spaceships). For the randomization to work we had to implement lots of boundaries according to the different possible radius, distances between objects, colors, illuminations, coordinates, etc.

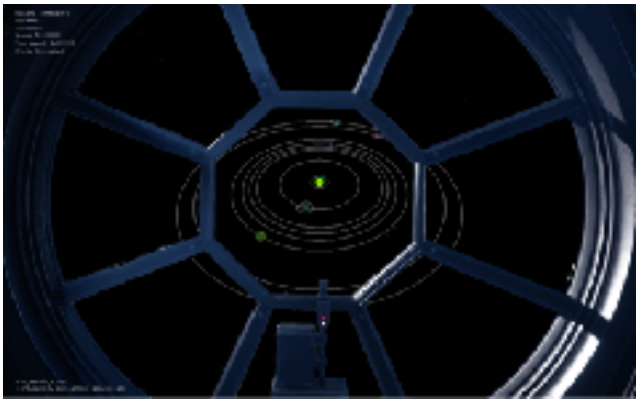
In our main file we either generate a random seed or use the one that the user wants us to use and we name that number “Region”, once having a region we start a waterfall where we create random systems that create random planets that create random… and so on, with this we allow the users to go to certain region whenever they want to (re)explore or share with other users the discovery

IV. USER INTERACTION

Movement			
w	Forward	i	Pitch Down
a	Left	j	Roll Left
s	Backward	k	Pitch Up
d	Right	l	Roll Right
q	Yaw Left	e	Yaw Right
,	Decrease spacecraft speed	.	Increase spacecraft speed
-	Decrease time speed	=	Increase time speed
View			
r	Real size planets	0 - 9	Point to solar system X in that region
o	Toggle orbits	x	Point to current* spaceship
z	Point to previous spaceship	c	Point to next spaceship
Misc			
b	Play previous song	n	Toggle play
m	Play next song	ESC	Exit simulation

IX. INTERFACE





IX. Conclusion

The computer graphics have a lot of applications and sometimes we don't recognize the real effort and power required for them to work, we achieved to develop a simulator that auto generates systems that can be ported to any devices that understand OpenGL 2 and we believe that this gave us a really big perspective for future developments.

Acknowledgment

We want this project to be part of the open source community and use it as a guide for their future developments, as well we want to thank the community for giving such useful snippets and quick answers on the internet, and particular thanks to Ryan Pridgeon for allowing us to read of one of his repositories and use some of his logic and later on, to implement our own technics.

References

1. A. Moises, “Graficas Computacionales TC3022.01” PDF Presentación de clase 2017
2. R. Pridgeon. (2015, Nov 10). Solar Systems [Online]. Available: https://github.com/RyanPridgeon/solarsystem
3. Design of a programmable vertex processor in OpenGL ES 2.0 mobile graphics processing units. Shen-Fu Hsiao; Po-Han Wu; Chia-Sheng Wen; Li-Yao Chen. Published in 2013 International Symposium onVLSI Design, Automation, and Test (VLSI-DAT). Pages: 1 - 4. Extracted from: http://0-ieeexplore.ieee.org.millenium.itesm.mx/stamp/stamp.jsp?arnumber=6533843
4. Real-Time Blur Rendering of Moving Objects in an Augmented Reality Environment. Zhao Yue; Li Jing jiao; Li Zhen ni; Ma Ji. Published in 2014 Sixth International Conference on Intelligent Human-Machine Systems and Cybernetics. Year: 2014, Volume: 2. Pages: 138 - 141. Extracted from: http://0-ieeexplore.ieee.org.millenium.itesm.mx/stamp/stamp.jsp?arnumber=6911467
5. Teaching an introductory computer graphics course using OpenGL. Daniel C. Cliburn. Published in Journal of Computing Sciences in Colleges. Volume 19 Issue 1, October 2003, Pages 102-103. Consortium for Computing Sciences in Colleges , USA. Extracted from: http://0-delivery.acm.org.millenium.itesm.mx/10.1145/950000/948751/p102-cliburn.pdf?ip=207.249.33.146&id=948751&acc=PUBLIC&key=AC2CD171D2BF46AB%2E76862172A10D4F54%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=936514782&CFTOKEN=31816344&__acm__=1494807774_47a21d7bbf8003258e2a4565ee7ad829
6. Application of advanced rendering and animation techniques for 3D games to softbody modeling and animation. Miao Song and Peter Grogono. Published in C3S2E '09 Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering, Pages 89-100. Montreal, Quebec, Canada — May 19 - 21, 2009. Extracted from: http://0-delivery.acm.org.millenium.itesm.mx/10.1145/1560000/1557640/p89-song.pdf?ip=207.249.33.146&id=1557640&acc=ACTIVE%20SERVICE&key=AC2CD171D2BF46AB%2E76862172A10D4F54%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=936514782&CFTOKEN=31816344&__acm__=1494807646_64a770509d670b9322e96cbe50f64b9b.
7. Modern OpenGL programming. Hongwei Li. Published in SA ‘14 SIGGRAPH Asia 2014. Article No. 12. Shenzhen, China — December 03 - 06, 2014. Extracted from: http://0-delivery.acm.org.millenium.itesm.mx/10.1145/2660000/2659473/a12-li.pdf?ip=207.249.33.146&id=2659473&acc=ACTIVE%20SERVICE&key=AC2CD171D2BF46AB%2E76862172A10D4F54%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=936514782&CFTOKEN=31816344&__acm__=1494807504_872d5e4c2145a96d657cd2640679a53f
8. https://www.nomanssky.com
9. https://www.models-resource.com
10. http://www.libpng.org/pub/png/libpng.html
11. https://www.opengl.org
12. http://devernay.free.fr/hacks/glm/
13. https://www.libsdl.org/
14. http://openil.sourceforge.net
15. https://math.stackexchange.com/questions/180418/calculate-rotation-matrix-to-align-vector-a-to-vector-b-in-3d