



by Sebastian Haushofer  
& Christoph Hochrainer

---

### Controls:

- W – move forwards
- S – move backwards
- A – move left
- D – move right
- Space - jump
- Left mouse – shoot red bullet
- Middle mouse – shoot green bullet
- Right mouse – shoot blue bullet
- Mouse wheel – zoom in/ out
- G – enter/ exit god mode
- F2 – show FPS and number of rendered objects on/ off (default off)
- F3 – wireframe mode on/ off (default off)
- F4 – nearest neighbor / linear texture mapping without mipmaps (default not used)
- F5 – nearest neighbor / linear texture mapping with mipmaps (default linear)
- F8 – frustum culling on/ off (default on)
- F9 – alpha blending on/ off (default on)

## Effects:

- Cel Shading - [https://en.wikipedia.org/wiki/Cel\\_shading](https://en.wikipedia.org/wiki/Cel_shading)  
It is used for rendering the golden projector in the first room and the golden temple in the final room. The number of different shading intensities can be set as uniform parameter in the shader.
- GPU Particle System – [https://tuwel.tuwien.ac.at/pluginfile.php/866185/mod\\_page/content/14/Particles\\_SS17.pdf](https://tuwel.tuwien.ac.at/pluginfile.php/866185/mod_page/content/14/Particles_SS17.pdf)  
There are two particle systems in our game. The first one is moving dynamically and is attached to the color canon. It shoots out particles from the canon in the same color as the bullet which is fired. It only shoots particles for a short amount of time when a bullet is fired. The rest of the time it won't do anything.  
The second one is static and directly underneath the golden temple. It constantly fires particles in fading colors. The colors of the generated particles are constantly changed via a Perlin Noise algorithm. The point light above this particle system always shines in the same color as the particles which are currently being emitted.  
Both systems run on the GPU using geometry shaders and transform feedback buffers. Each particle has the basic properties position, velocity, color, lifetime, size and type. The type specifies if it is a regular particle or a particle which generates more particles. The rest should be self-explaining.
- Spotlights - <https://learnopengl.com/#!Lighting/Light-casters>  
There are two spotlights. One is attached to the camera and moves along with you and the other one moves on a straight line and also casts shadows. Both lights have an inner and outer cone to avoid a single hard light intensity change from completely lit to unlit. Instead the light's strength is interpolated between these two values.
- Shadow Maps - [https://www.cg.tuwien.ac.at/courses/CG23/slides/tutorium/CG2LU\\_Tutorium.pdf](https://www.cg.tuwien.ac.at/courses/CG23/slides/tutorium/CG2LU_Tutorium.pdf)  
In our project we have two hard coded shadow maps.  
The first one is for the directional light and the second one is for the moving spotlight in the second room. The shadow map for the directional light is achieved by rendering only the depth of the scene onto a texture with an orthographic projection from the light's view direction. The spotlight uses a perspective projection matching its light cone to create a matching shadow map. This is done every update/ frame. The implementation is similar to the CG-UE slides mentioned.
- Projected Textures – same slides as shadow map  
We have one projected texture in the main room which is “attached” to the beamer under the big ball and needs a third shadow map. It rotates and updates also every update/ frame. For projection texture we use the same technique as shadow mapping but instead of multiplying the unlit areas with a percentage we add a percentage of a picture/ texture to the lit areas of the scene. The implementation is nearly the same as for the shadow map but an additional texture has to be loaded into the shaders. Again we have implemented it similar to the explanation in the slides mentioned above.

## Meeting the requirements:

The player plays the game in the first person perspective. Therefore the camera is freely movable. Of course you can't walk up in the air. (Unless you enter god mode which allows you to fly around the world.) But other than that you can move around in the x-z plane (even with little height differences) and look around left/right and up/down with the mouse as you like. Additionally you can jump to move the camera.

The enemy robots have an AI script which makes them follow you if they currently have any color attached to them and are close enough to you. Basically they will always turn straight towards you and move towards you in a straight line. You can recognize that they are following you if they are lifting their arm up and down. This means that they are trying to paint you in their color. If they are not following you or are neutral they will just walk along their default target points and hold their arm still. Besides the color canon is attached to the camera and moves along with you as you walk through the world. The canon is animated so that when you shoot it rotates a little bit up and then down again. There is also a rotating projector and a moving spotlight.

The robots, the color canon and almost all objects of the world have textures attached to them. The models have been created by us (except for the projector and the spotlight) using blender and exported as collada .dae files. Even the entire world has been modelled and textured using blender. Our model loader which uses the assimp library does not only load the 3d data, it also attaches the appropriate collision shapes to the models. Apart from that it can be used to set robot spawn and target points, shaders, mass and more settings for the objects directly in blender. Basically we have written an importer to make blender our little level editor.

The "temple" in the last room and the projector in the first room both have a golden material assigned to them and are rendered with cel shading. The robots also have a different material assigned depending on how they are currently colored. For the robot's fragment shader the output color is a combination of the robot's texture and material.

There are four light sources. First of all there is a weak directional light simulating the moon which casts shadows. Besides there is a point light under the golden temple in the last room and a spotlight which is attached to the camera and is responsible for the main lighting. These two light don't cast any shadows. Finally there is a spotlight in the second room which casts shadows and moves along a straight slide lighting the different layers and robots from slightly above in the second room.

The frustum culling is implemented via the bullet library where the already existing collision shapes of the 3d objects for the collision detection are used. An additional frustum collision shape for the camera has been generated as a trigger to detect which objects should be rendered each frame. Frame buffer objects are used for the shadow maps and for the projected texture. Mipmapping and texture sampling settings can be changed during runtime (see Controls). Blending is used for the particle system to make the particles fade out as their lifetime decreases. This can also be turned off and on.

### Important Notes:

For performance reasons the different effects are only active and calculated when the player is in the right room. For example the projected texture in the first room will only be visible when the player is in the first room and the spotlight casting shadows will only be active when the player is in the second room. To switch between these effects and detect when the player is entering a new room bullet collision triggers have been used. The triggers are invisible and only fired when the player actually walks through them.

**This means if you fly from one room to another in god mode through the sky the effects won't get activated!**

However all special effects can be already seen in the first room. (Assuming you have already collected the color canon above you to fire bullets.)

**There is a configuration file located under bin/colorsplash/ColorsplashConfig.cfg where you can adjust several parameters such as brightness and mouse sensitivity.** You can also activate gamma correction and full screen mode in this file.

## Features:

We have integrated bullet as physics library to our project and the collision system works well. The camera has a capsule collider attached to it and you collide with the walls/ robot enemies and can jump up the different platforms.

When you start the game the robots are already colored differently by default and will mind their own business. You need to shoot them with the appropriate colored bullet to neutralize them and to stop them from following you if you get too close to them. If a robot is green for example you need to shoot it with a green bullet. If you hit it with another color let's say red for example, it will absorb that color. In this example the robot will become yellow (red + green). Per basic color (red, green and blue) the robot has absorbed it will move faster. You can "hide" from the robots by jumping up the platforms or walking behind another object.

Additionally we have added game sounds for different situations in the game. For example when you walk, a bullet is fired, you lose or win or when a robot starts/ stops following you a sound is played. Apart from that we have implemented a skybox of the night sky which gives the game a more realistic and better look.

Finally we have also programmed text rendering to show when the player has won/ lost and to display the framerate and amount of rendered objects if requested.

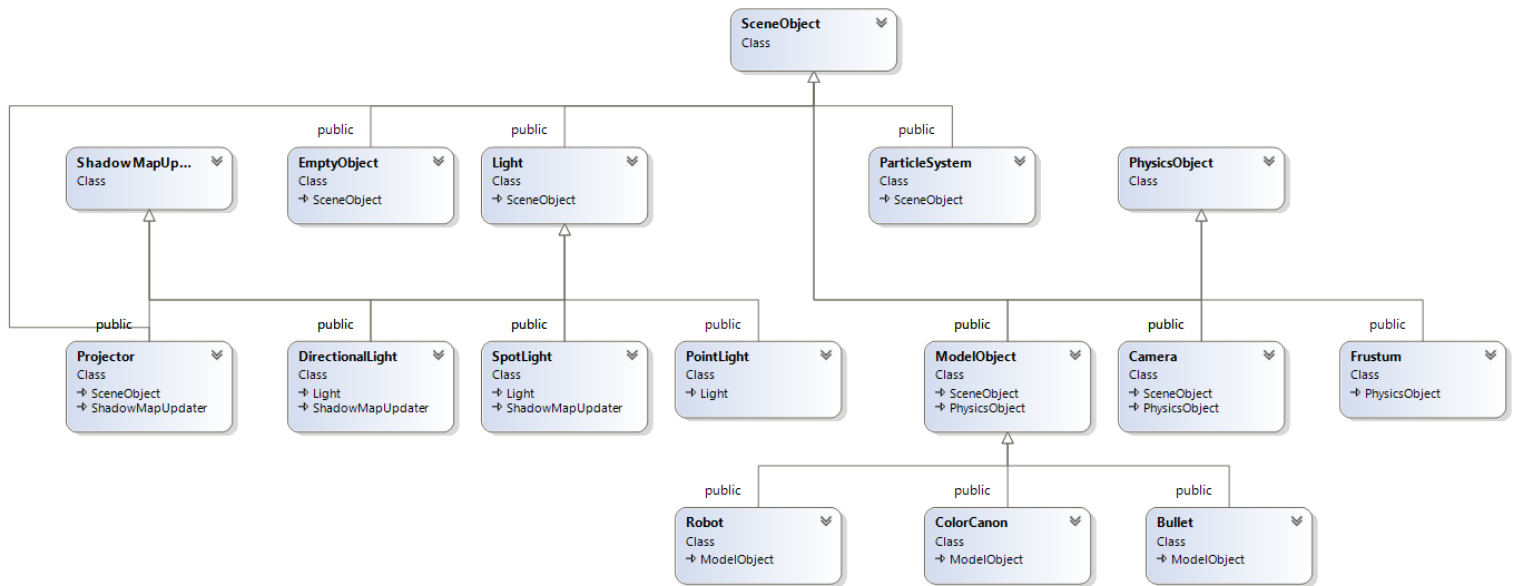
## Libraries and Tools:

- glew: graphics library for Open GL  
<http://glew.sourceforge.net>
- glm: math library for Open GL and graphic calculations (matrices, vectors, etc.)  
<http://glm.g-truc.net/0.9.8/index.html>
- glfw: window manager for window creation and managing  
<http://www.glfw.org>
- assimp: model loader  
<http://assimp.sourceforge.net>
- freeimage: texture/image loader  
<http://freeimage.sourceforge.net>
- bullet: physics engine  
<http://bulletphysics.org/wordpress>
- irrKlang: audio library  
<http://www.ambiera.com/irrklang/>
- freeType: font rendering library  
<https://www.freetype.org/>

## Implementation Notes:

Almost all knowledge which was required to implement the features mentioned above has been gathered from <https://learnopengl.com> which is a great OpenGL tutorial with a lot of examples. Some of the code from this tutorial has been taken as a basis and improved and tweaked to our needs (for example the camera and model loader).

In our engine everything making up the scene is a SceneObject. Here is an UML class diagram of the inheritance structure from it:



Each SceneObject has an optional list of children SceneObjects and an optional parent SceneObject which allows us to update and draw the scene recursively. The head of our scene graph (actually it is a scene tree) is an EmptyObject at origin without any rotation and a scaling of one. From there on downwards all coordinates and rotations are relative to the parent which allows us to attach objects relative to other object easily. For example the spotlight and the color canon are set as children of the camera which again is set as a child of the EmptyObject which is the scene graph's root.

Apart from that there are quite some manager classes which are responsible for creating and deleting the different kinds of shaders, settings, materials and so on used in our game.