config.cfg Usage

In the config.cfg you can set different parameters for the game, such as resolution or fullscreen. For testing without Enemies you can set *numEnemies=0*. For testing with a invincible player you can set *godMode=1*. All parameters are described in the config file, so hopefully it is clear what they are for.

Gameplay

Controls: W,A,S,D to move ,spacebar to jump, mouse to aim and right mouse button to shoot

The Player controls a block that moves around in a giant block and has to shoot the red colored blocks, before they shoot him. The player and the enemies both die with one shot, so if the player gets hit he explodes and the game is over. If he reaches a certain amount of enemy kills (default=30) he wins the game and gets celebrated with fireworks. After winning the game, you can look up your needed time in the console

We assured that our game is playable by implementing a collision detection, to force the player to move inside our world and make the shooting mechanic of our game work. We have real 3D gameplay, because the 3D world allows movement that wouldn't be possible in 2D. Sometimes he has to jump on something to be able to shoot at the enemies. The 3D also changes the way u can use cover, since enemies could shoot from a higher position.

Effects

Spotlights

References:
http://www.ngreen.org/?p=527

We implemented the part, that was important for the spotlight-effect, since the rest was already implemented for the point light from submission 1. The shader needed the direction and the range of the light, to know how far and which direction the spotlight should shine.

The spotlight is positioned in front of the player and has the same direction as the player is looking.

Shadow Maps with PCF

References:
http://ogldev.atspace.co.uk/www/tutorial23/tutorial23.html
http://ogldev.atspace.co.uk/www/tutorial24/tutorial24.html
http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/
http://http.developer.nvidia.com/GPUGems/gpugems_ch11.html (for PCF)

We created a depth texture, which is rendered from the position of the light in its direction. Because we are using a spotlight we used a perspective projection matrix. The content of this texture can be shown by setting *showShadowMap=0* in the config file. The contents of this texture are then used to check if vertices shown by the camera are in the shadow or not.

We implemented our own PCF algorithm based on the description of the reference for PCF. The shadows are then rendered for the spotlight.

You can see the shadow well, when you direct the spotlight at the teapot. We use the backfaces for the shadow, which is the reason for the shadow looking like it is. We did this, becaus when we use the frontfaces instead we get alot of artifacts.

Cel Shading

References:
https://en.wikipedia.org/wiki/Cel_shading
http://prideout.net/blog/?p=22

In the fragment shader, 8 “buckets” are defined for the [0, 1] range. A function “celShade” is defined, which takes a float of that range as input and returns the base value of the bucket in which the input value belongs.

The diffuse and specular coefficients get passed through this function, resulting in clear-cut transitions between darker and brighter areas.

In addition, the transition between the inner and outer cone of the spotlight is cel shaded as well, to keep the cel shading look consistent.

+ Contours (backfaces)

References:
https://en.wikipedia.org/wiki/Cel_shading

The back faces of all instances (including particles) are drawn slightly bigger wire frame mode with black ink.

In the vertex shader, every vertex gets displaced by 0.005 points along its vertex normal to make the object slightly bigger.

By drawing only the back faces only the contour remains after the z test.

Due to drawing the wire frame, the contour remains its thickness throughout the scene.

CPU-Particle System (+Instancing)

References:
http://ogldev.atspace.co.uk/www/tutorial33/tutorial33.html

A Particle struct captures values like position, direction and traveled distance. In each update cycle the particles are updated, yielding a new model (transform) matrix. If a particle traveled a certain distance at this point it will be discarded.

In the draw method the model (transform) matrices of each particle are written into a VBO.

The “model” matrix of the instancing shaders has been changed from a uniform input to a regular one. Thanks to “glVertexAttribDivisor” and “glDrawElementsInstanced” all particles are drawn within a single call.

Projected Textures

References:
https://en.wikipedia.org/wiki/Projective_texture_mapping
http://www.ozone3d.net/tutorials/glsl_texturing_p08.php
https://developer.nvidia.com/content/projective-texture-mapping

A texture gets projected from the player into the world based on the mouse position to assist with aiming.

A (perspective-) projection and view matrix are created from the player in the “aim” direction. These matrices get multiplied with the model matrix of each instance that gets drawn. The resulting vertices are in a [-1, 1]^3 range, while the texture coordinates are in a [0, 1] range. To account for this discrepancy, the texture-projection MVP matrix gets multiplied with a bias matrix, so that the resulting coordinates in will be in a [0, 1] range.

In the vertex shader the vertices get multiplied with the texture-projection MVP matrix.
In the fragment shader the texture gets sampled with the xy coordinates of the resulting vec4. To account for the perspective projection, the coordinates are divided by their homogenous component first.

Finally, the color of the fragment gets multiplied by the sampled color (if available).

Animated Objects

n/a

View-Frustum-Culling

n/a

Transparency

References: http://www.opengl.org/archives/resources/faq/technical/transparency.htm

Particles are drawn with an alpha value of 0.5 to distinguish them from bullets and enemies.

Experimenting with OpenGL

VAOs / VBOs are used extensively throughout the project.
A FBO is used to draw the depth values from the point of view of the light source into a texture.

Illuminated/Textured Objects

We use a movable spotlight, that shows in the direction the player is facing. Because our light source is movable everything except the player, who is behind the light, can be lit. The world, the enemies, the bullets and the teapot all use the same texture with different colors. We did this to give our game the simplistic look we wanted. Use the *complexTexture=1* option to use a different texture.

Tools for Model Creation

Blender

Additional Libraries

Assimp - http://assimp.sourceforge.net/
Bullet - http://bulletphysics.org/