Gameplay:

The player can move by using the wasd keys and jump by pressing space.

Effects:

Shadow Maps:

For Shadow Maps two separate passes are rendered, the first pass renders into the depthbuffer from the view of a directional light and then creates a texture. The second pass is a normal renderer. The fragment shader now uses the shadowMap and calculates if the objects are shadowed or lighted. To account for the movement of the player, the light is given a virtual position to render the shadowMap for objects near the player.

Source: https://www.learnopengl.com

GPU Particle System (Compute Shader, Instancing):

Particles are implemented as small yellow cubes suggesting fireflies. The compute-shader calculates random values for their movement, but keeps them in a fixed range around the spawning point. Every work group has a starting position as a n instanced attribute for that particular work group. This effect was implemented by using the following links as reference and using a mixture of ideas of the two implementations:

https://www.cg.tuwien.ac.at/courses/Realtime/repetitorium/rtr_rep_2014_ComputeShader.pdf

http://web.engr.oregonstate.edu/~mjb/cs557/Handouts/compute.shader.1pp.pdf

Cel Shading + Contours (Edge Detection):

A standard toon shader with 3 levels is being used. The diffuse value is being calculated by level numbers. A 5 by 5 Laplace-filter is used with 24 as the center detecting the edges. At first only the player character is being rendered into a framebuffer. The points with a higher summed-up color-value than 0.9 after the laplace-filtering are set to 0 (black), the others are set to transparent. After the final scene was drawn the edge detected image is being rendered over the whole scene. This feature was implemented by using the following link as a reference for the laplace filter and the tips of feedback talk 2:

http://r3dux.org/2011/06/glsl-image-processing/

Complex Objects:

Assimp is being used as the model loader library. One of the complex objects is, for instance, the starting island.

View-Frustum-Culling:

View-Frustum-Culling is implemented by checking the boundaries of the physx objects if the object is in bounds of the frustum.

Transparency:

A plant is visible that has a quad on the top. Parts of it are transparent, that’s why it looks like a leaf.

Lighting and Textures:

A directional light is being used. FreeImage is being used to load the image-files for the textures.

Models were created using Maya.

Experimenting with Open-GL:

A variety of Opengl functionalities and containers are being used. Including Vertex-Buffer-Objects, Vertex-Array-Objects, Frame-Buffer-Objects etc.

Resources:

http://www.assimp.org/

http://freeimage.sourceforge.net/

http://www.lighthouse3d.com/tutorials/

www.learnOpenGL.com.

http://www.autodesk.de/products/maya/overview

http://learnopengl.com/#!In-Practice/Text-Rendering