Bouncing Blocks

Controls

General

Description	Button
Reset	R
Debug	T
Camera Movement (debug mode only)	Arrow keys
Pause	P
Continue	Space
Wireframe	F3
partyMode	F7
Frustum culling	F8

Players

Player name	Up	Down	Left	Right
Blue	W	S	A	D
Red	I	K	J	L
Green	Arrow up	Arrow down	Arrow left	Arrow right
Yellow	Numpad 8	Numpad 2	Numpad 4	Numpad 6

Effects

Shadow Mapping

We use shadow mapping with a directional lightning on all objects. For this, we first render a depthmap to a framebuffer (seen from the light source) and then compare in a second pass the z values of the objects to see if the current fragment is hidden. If this is the case, the fragment is rendered darker. To create a smooth transition, we sample the shadowmap four times. A very helpful resource for accomplishing this task was found in:
http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/

Bloom

We implemented Bloom as follows: First the scene is rendered to a texture in a FrameBuffer which is four times smaller than the normal resolution. Then it is processed with a Gaussian Filter through two steps. The first substracts a certain amount from the color (to only leave the brightest spots visible) and blurs the texture horizontally. The second blurs the result vertically. These two steps also work through rendering to textures bound to FrameBuffers. Then, in the final rendering stage, the amount of the according fragment on the downsampled, blurred image is multiplied by the factor of 9 and added to the normal color value to achieve the Bloom effect. The value of the factor was found out through try and error and 9 was considered best.
Helpful information for accomplishing this task were how to use framebuffers:
http://antongerdelan.net/opengl/framebuffers.html
http://en.wikibooks.org/wiki/OpenGL_Programming/Post-Processing
and how to gaussian blur:
http://www.gamerendering.com/2008/10/11/gaussian-blur-filter-shader/

Cel Shading

We use cel shading for all objects. In order to keep the details of our self painted textures we decided to use cel shading for lighting only. We clamp the value of light intesity via a 1D-Texture.

Contours

Countours are drawn with the back faces of the objects. We use interpolated normals to decrease the size of the front polygons. Those normals are calculated in a pre-processing step (You can find more on that in section “Google Buffer Protocol”). We decided to decrease the size of the front faces, so that there are no overlaps of faces when colliding objects.
Helping resource: http://www.sunandblackcat.com/tipFullView.php?l=eng&topicid=15

Particle System

We use a CPU based Particle System. We took the advice on this page:
http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/particles-instancing/
The particles are behind the play field. You see there are colored points popping up from nowhere. Use the debug mode to navigate to get a better look.

Complex Objects

We have the cow mesh, the laser mesh as complex objects in our scene. All objects are loaded via assimp or to be precise, have been loaded with assimp, but are now loaded as binaries with google buffer protocol.

Animated Objects

We have one hierachical object: It consists of the player cube, which is connected with the stone cylinder which is connected with the cow. The matrices are multiplied with one another and the cow itself is spinning as well.

View-Frustum-Culling

For Frustum culling we used a rather old tutorial: http://www.crownandcutlass.com/features/technicaldetails/frustum.html The implementation was not too hard, even though we should note, that the flame thrower with its particles does not get culled, when it is out of view.

Transparency

The text of the gui as well as our flame particles have transparent and semitransparent regions. It is not possible to turn them on and off.

Experimenting with OpenGL

In our game we use:

- Vertex-Buffer-Objects (VBO)
- Vertex Array Objects (VAO)
- Frame Buffer Object (FBO)
- Mip Mapping
- Textur-Sampling-Quality (Trilinear Filtering)
- Frustum Culling
- Wireframe (On/Off)

Those effects can’t be turned off.

Used Libraries

Glew&Glfw	Using OpenGL & Window management
Assimp	Loading meshes
OpenAL	Sounds
Bullet	Physics
GLM	Math
Google Buffer Protocol*	Saving and loading binaries

* We are using Google Buffer protocol, because our loading time for meshes increased drastically after we started calculating interpolated normals of all vertices. The protocol buffer helps us, to save all calculations and mesh data in binary files. Those can be read way faster.

Used Tools

Blender	All meshes were created in Blender (except of the cow)
Gimp	We used gimp to make some textures tileable (like the bricks texture)
Inkscpe	Only the logo
Audacity	Record sounds

Special features

We have a party mode, in which all players can rotate in all directions and other sounds are played. Have fun!

All assets, except of the cow, are selfmade. The sounds are recordings of our beatuiful voices and the textures are painted with water colors. If you look closely, you can even see brush strokes in some textures. You probably ask yourself, how we managed to create such beatuiful paintings. The answer is, we watched a lot of Bob Ross while painting.