Magnosphere, 2nd submission

Documentation

Gameplay:
Click mouse left/right to activate or deactivate magnetism. Roll around with WASD. Try to push the triggers (red cubes) to open doors, so that you can reach the goal with the big sphere and win the game.

Complex Objects:
We use assimp to load collada-files into our game. The objects are mainly spheres and boxes, but to show that we can import all kind of complex objects, we also included the CG2-rubberduck©. To provide a curved surface (and to make the leveldesign more interesting…) there is also a pipe, which was constructed using a simple boolean operation on two cylinders.
With exception of the rubberduck, the level was entirely self-modeled in blender.)

Animated Objects:
The rotation of the windmill is a hierarchical animation. (Parent and children are defined in Blender.)

View Frustum Culling:
View frustum culling is implemented via bounding boxes, because these are best suited for our (mainly cubic…) objects. This was implemented with help of the lighthouse-3D tutorial and also the tutorial by Mark Morley. (http://www.lighthouse3d.com/tutorials/view-frustum-culling/ and http://www.crownandcutlass.com/features/technicaldetails/frustum.html)

Transparency:
The ramp that's going up to the pipe is our (only) transparent object.

Experimenting with Open GL:
FBOs, VBOs, and VAOs are used in the game (in the SceneObjects, for the Effects etc.)

Lightning/Illumination and materials:
We have one big point light source in the scene, all objects in the scene are illuminated with this bulb light. All the objects use a simple texture shader.

Features:
-Some hand-drawn textures!
-Textured Objects
-Usage of PhysX to simulate the physics, to do collision detection, to generate/load bounding objects right from the meshes we have in the collada files.
-Object loader (loading colladafiles, based on assimp). Game Logic for Triggers/doors works automatically if the objects are given the right names in the .blend file before exporting it as collada-file.
-Illumination by point light, texture shading
-Freely moveable camera
-Magnetism realized via distance between the spheres
-Clever logic to assign action objects to triggers.
-Controls implemented with polling

-Creating our own view-matrix (without glm::lookAt)
-Glow effects and particle systems

Effects:

Glow (1P): The glow effect was implemented based on what was said in the repetitorium and the article from GPUGems(http://http.developer.nvidia.com/GPUGems/gpugems_ch21.html) as additional reading. The bluring in the shader for the alpha-texture was taken from DelphiGL(http://wiki.delphigl.com/index.php/shader_blur2), they use one shader for the horizontal and the vertical pass, the direction can be used via the uniform uShift – this was quite an elegant way to handle the two passes. The blending in the shader is a simple additional alpha blending, which is only executed when the uniform "blend" is set to 1.
If you want to see only the alphatexture, you can set the variable "colorRender" to "colorAlpha" (last commented line in the shader, was handy for debugging). The alpha texture for each object is saved in the alpha-channel of each .png-texture.

GPU-Particle System (1P): The GPU-based particle System was made with help of this tutorial: http://ogldev.atspace.co.uk/www/tutorial28/tutorial28.html, although some changes had to be made because the tutorial was for OpenGL 4.0+. (For example, the command "glDrawTransformFeedback" is not existent in OpenGl 3.0, you have to use "glBeginTransformFeedback" and also use a query to keep track of the particlecount).

Omni-directional shadow maps with PCF (2P):
Because we use just one point light to illuminate the whole scene, we're using omni-directional shadow maps. The shadow mapping was implemented with the help of Peter Houskas slides for the RTR Repetitorium (http://www.cg.tuwien.ac.at/courses/Realtime/repetitorium/2011/OmnidirShadows.pdf) with a slight modification: We're using a cubemap with six render calls instead of the geometry shader – it was not possible (even after approx. 2 weeks of debugging) to get the geometry shader working with the cube map. This causes of course a loss of performance.

Modeling: The modeling and UV-Mapping was done in Blender, the only object that's not self-made is the rubberduck. The textures were created with photoshop.

Additional libraries used:
Nvidia PhysX (https://developer.nvidia.com/physx)
Assimp (http://assimp.sourceforge.net/)
FreeImage (http://freeimage.sourceforge.net)

Step-by-step walkthrough:
→ Touch the first trigger in first room, small door opens.
→ Roll out of the room, pass the bridge and hit the trigger at the other side of the thorns.
→ Small door next to the first room opens, roll through and up the ramp. Push the next trigger up there, the big door opens and you can get the big sphere out of the first room.
→ Roll outside of the second room to the part of the level with the pipe. Roll up the ramp, pass the pipe and hit the next trigger.
→ Another door opened and you can roll the second sphere to the goal-area next.
→ Hit the big goal-trigger with the *big* sphere → WIN!