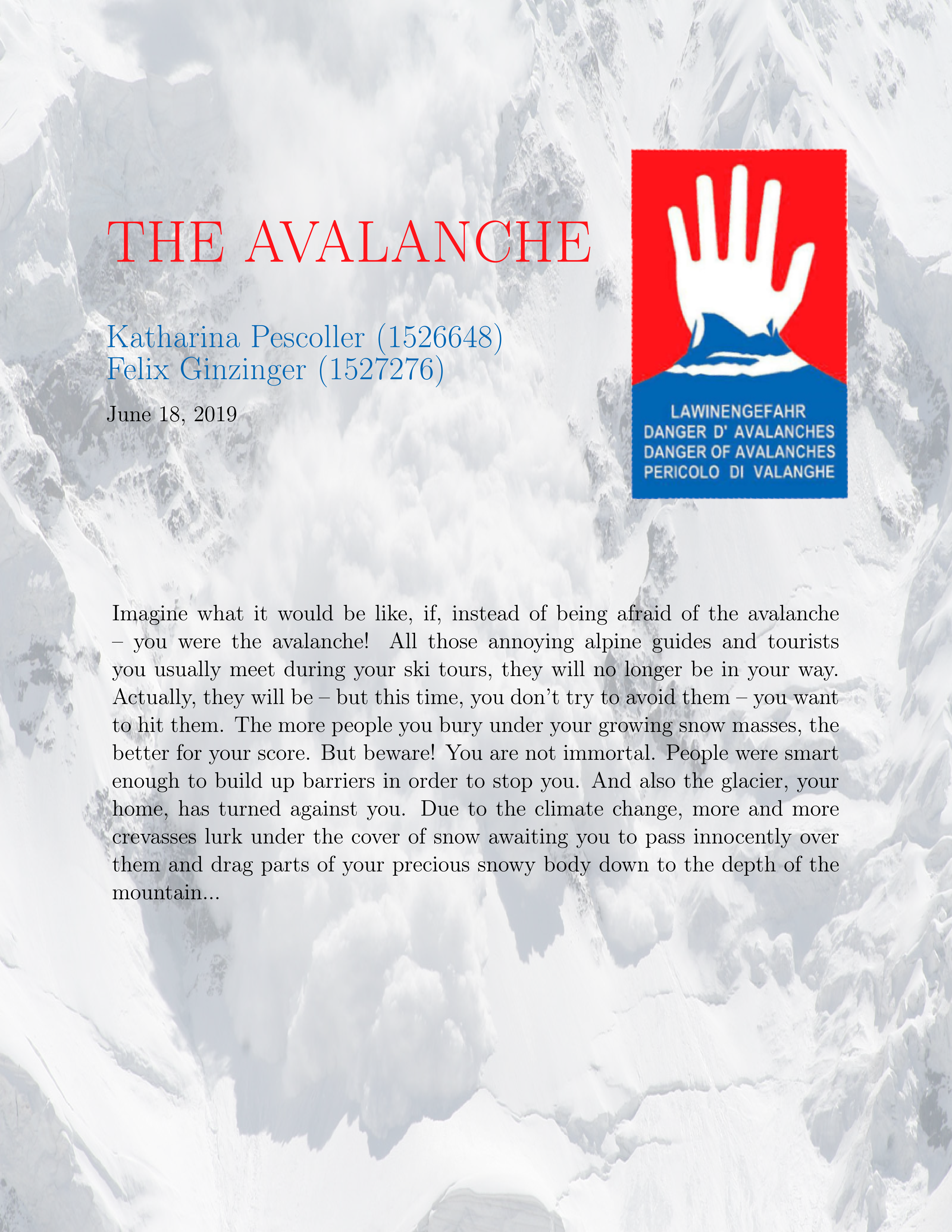

THE AVALANCHE

Katharina Pescoller (1526648)
Felix Ginzinger (1527276)

June 18, 2019

Imagine what it would be like, if, instead of being afraid of the avalanche – you were the avalanche! All those annoying alpine guides and tourists you usually meet during your ski tours, they will no longer be in your way. Actually, they will be – but this time, you don't try to avoid them – you want to hit them. The more people you bury under your growing snow masses, the better for your score. But beware! You are not immortal. People were smart enough to build up barriers in order to stop you. And also the glacier, your home, has turned against you. Due to the climate change, more and more crevasses lurk under the cover of snow awaiting you to pass innocently over them and drag parts of your precious snowy body down to the depth of the mountain...

1 Implementation

1.1 Gameplay

1. Playable: key callbacks, mouse movement callbacks for the player movement. Barriers are randomly set within a specified range around the player. The player moves onto a terrain created by midpoint displacement whose size and roughness can be set arbitrary by adjusting few parameters - not by the user, but by us. Also the amount of barriers can be changed easily by changing a single parameter - very smooth :)

2. 3DGeometry: we implemented our own object loader, no lib, for .obj files. We only provide rocks and one type of skier as barriers. The player is a cube with particles crawling around it.

3. Win/Loose: Player has to gain as many points as possible in given time frame.

4. Intuitive Controls: w, a, s, d as usual. mouse movement for view and movement direction changes.

5. Intuitive Camera: camera follows player

6. Illumination and Textures: Barriers and Player are illuminated using Physically based Shading Cook-Torrance model **??**. The terrain has a procedural generated texture **??**. Furthermore we implemented lens flares 2 and a simple texture mapping using a .png image for the background.

7. Moving Objects: our player moves, all barriers do not move by their own but as they do have a collision shape they are forced by gravity to fall down onto the terrain.

8. Documentation: see source code

9. Physics Engine: we implemented collision detection with changing points, velocity and size based on collision. We create a PhysX heightfield out of our terrain height map on which the player can move around freely.

10. Adjustable parameters: we provide a settings.config file where you can change width, height, refresh rate, fullscreen mode, gamma value for brightness on monitor and title of the window. Furthermore width, height and fullscreen (toggle with F1 / ESC) can be changed during runtime.

11. Heads-up Display: we provide a HUD for information about the current state of the game (points / time) and one for the initial information (how to start the game, how to toggle fullscreen).

1.2 Effects

1. GPU Particle System using Transform Feedback: particles are emitted from the center of our player and crawl around it forced by gravity to fall down. Their velocity / direction depends on the players movement. We use two shaders for the effect: the first one (denizanGucer) consist of a vertex and geometry shader emitting the particles based on random values obtained from textures. The second one (renderParticles) is used for the rendering part and hence consists of a vertex, geometry (billboard creation) and fragment shader.
   For the implementation of Transform Feedbacks we read through:

   - https://www.zerosumfuture.com/blog/2018/10/7/a-modern-particle-system and
   - https://smunix.github.io/ogldev.atspace.co.uk/www/tutorial28/tutorial28.html

   Our implementation extends those tutorials with respect to the emission and behaviour of the particles, which, in our case, correspond to the players movement.

2. Lens Flares : For the lens flares we read the blog

   - http://john-chapman-graphics.blogspot.com/2013/02/pseudo-lens-flare.html

Our implementation considers sun location as an extension to the one presented by John Chapman, i.e. halos and ghosts are formed around the sun location instead of the center of screen and also they are only shown when the viewer looks straight at the sun.

3. Physically Based Shading (Cook-Torrance) : For the implementation of this we read through the original paper [1] and used the web page

   - http://www.codinglabs.net/article_physically_based_rendering_cook_torrance.aspx

   However we only consider one light direction (the straight direction from the sun) as we did not implement any environment maps. We provide a file material.dat where materials are defined and in the subsequent used for skier, rock and player.

4. Tessellation from height map: For the terrain-Tesselation we read the follwoing tutorials:

   - http://codeflow.org/entries/2010/nov/07/opengl-4-tessellation/
   - http://ogldev.atspace.co.uk/www/tutorial30/tutorial30.html and
   - http://ogldev.atspace.co.uk/www/tutorial31/tutorial31.html

   In our implementaion the amount of generated polygons depends on the distance to the camera. The Tesselation is done by two newly added shader stages: the Tessellation Control Shader (terrain.cs) and the Tessellation Evaluation Shader (terrain.es). We distinguish between three different distances to the camera / level detail which will be drawn. In order to change the height of the newly generated polygons, which is needed in order to gain a better impression of the terrain, a heigthmap-texture is loaded to the Tessellation Evaluation Shader and the new height value of the polygons is extracted out of it. To visualize the tesselation result better in the game, it is possible to toggle between wire-mode with the button "F1".

5. Procedural textures: Since we weren't really happy with the texture of the terrain we decided to use a procedural texture for the terrain in order to simulate a terrain which looks like an glacier environment. For this we added three functions to the Fragment shader of the terrain: one which generates random numbers, one that generates Perlin Noise (generates a texture out of adding multiple times Perlin Noise with different parameters and one time the noise function in order to apply a little bit the snow-like structure). The Perlin Noise is generated one time out of the xz-coordinates of the terrain to gain the nice cloudy structure and one time only with the height information in order to get fine lines along the steeper parts of the terrain hills. We read through the following webpages for help:

   - http://www.science-and-fiction.org/rendering/noise.html for random numbers
   - https://gpfault.net/posts/perlin-noise.txt.html for Perlin Noise

2 Additional Information

1. We use one single directional light source (the sun) which affects game objects (cook torrance model) and terrain (phong).
2. Additional library: PhysX 4.1 for collision detection (https://developer.nvidia.com/physx-sdk)
3. Implemented object loader inspired by (http://amin-ahmadi.com/2017/01/04/how-to-read-wavefront-obj-files-using-cqt/)
4. We downloaded the objects from https://www.yeggi.com.

References

[1] Robert Cook and Kenneth E. Torrance. A reflectance model for computer graphics. *ACM Trans. Graph.*, 1:7–24, 01 1982.