

# Kitty Madness

Administrative	
Group Name / Game Name	Kitty Madness
GitHub Link	<a href="https://github.com/JakobLindner/cgue20-kittymadness">https://github.com/JakobLindner/cgue20-kittymadness</a>
Students	Huber Marlene, 01601103 Lindner Jakob, 01634393
Genre	3D Platformer/Adventure
Goal	Throw items into the lava, while not falling in yourself, to advance to the next level.

Gameplay	
3D Geometry	We used an .obj object-loader (see <i>List of Libraries</i> ) and self-made/downloaded models.
Playable	Fluffy can roll around surfaces and jump. By pushing objects into the lava, it earns points to win, but dies when falling in itself.
Advanced Gameplay	Level-logic (Tutorial, Level1 and Level2), can jump on things, bounciness of objects, objects with different point values, jump higher when key is pressed down longer,...
Min. 60 FPS and Framerate Independence	Used delta-time - PhysX always advances exactly 1/60 independent of framerate.
Win/Lose Condition	Win: score enough points by throwing things into the lava Lose: fall into the lava
Intuitive Controls	See <i>List of Inputs</i>
Intuitive Camera	See <i>List of Inputs</i>
Illumination Model	Normal vectors were done in Autodesk Maya or came with the 3D-files. Material parameters are provided through a texture combining ao, roughness and metal.
Textures	All objects have UVs and textures attached. Textures were either given with models or made via Substance Painter.
Moving Objects	Fluffy moves and by doing that, it can push/move other objects around.
Documentation	
Adjustable Parameters	Screen resolution, fullscreen-mode, refresh-rate, mouse-sensitivity and VSync can be adjusted via the settings.ini (everything else – see <i>List of Inputs</i> )
Collision Detection (Basic Physics)	Fluffy (dynamic actor) can stand on objects and cannot jump/move through walls, the ceiling and other dynamic objects.
Advanced Physics	Fluffy is moved by applying forces in the given direction and it can move/push other dynamic objects around. When something hits the lava, a corresponding particle system spawns in that place.
Heads-Up Display	Alpha-blending is used. The HUD shows the current level, score in numbers and as a score-bar or a win/lose/loading-screen.

Effects	
Advanced Modelling	
GPU Particle System using Compute Shader	<p>We used a compute shader and a shader program consisting of vertex, geometry and fragment shader to realize this particle system. The compute shader keeps track of the global particle count via an atomic counter which is checked when particles are added. It de-spawns particles when their “timeToLive” is over and therefore frees up computing space for new particles. There are two shader-storage-buffers for velocity and position respectively to read/write the in- and outputs which update those values for each particle in the buffers. Initially, positions and velocity are generated via random numbers. The particles are drawn via VAOs, starting in the vertex shader which passes the values on to the geometry shader, which then generates a billboard quad for each particle; its size in relation to its time to live (getting smaller over time). In the fragment shader a texture is applied to this quad and the color as well as the alpha are manipulated with functions in a similar way (getting lighter and more yellow over time).</p> <p><i>Note:</i> See <i>List of Inputs</i> for purposely triggering particles. There are particles spawned at random throughout the game all over the lava – this is done via a random generated countdown corresponding to the delta time.</p> <p><a href="https://tuwel.tuwien.ac.at/pluginfile.php/1721131/mod_page/content/37/GPU_Particles_SS18.pdf">https://tuwel.tuwien.ac.at/pluginfile.php/1721131/mod_page/content/37/GPU_Particles_SS18.pdf</a></p>
Animation	
Vertex Shader Animation	<p>This effect is only applied in the lava’s vertex shader. To get a smooth, but not too regular, waveform we implemented a function that displaces the x and z position of the plane. To do so, we calculated this new position for the actual vertex and two additional virtual points (not actual vertexes from the object). After the position displacement we take all three points for the calculation of the new normal knowing that those three points have to be planar. The fragment shader only applies a texture and makes Phong light calculations to make the new normal visible.</p> <p><a href="https://tuwel.tuwien.ac.at/pluginfile.php/1721131/mod_page/content/37/Animation_SS18.pdf">https://tuwel.tuwien.ac.at/pluginfile.php/1721131/mod_page/content/37/Animation_SS18.pdf</a></p>
Texturing	
Environment Map	<p>We load the map via an HDR-loader (see <i>List of Libraries</i>) and generate the cube-map from the equirectangular image. The cube-map is used in the PBR’s diffuse and specular light calculations (IBL). An irradiance-map, a prefiltered-map and a BRDF-texture are calculated before the actual rendering in their own shaders. For the prefiltered-map, mipmapping is used and the different levels are applied according to the roughness of an object. To see the difference – have a look at the “Coffee Table” compared to any other object (e.g. backside of the blue chair). For this, we implemented an additional calculation for the Fresnel-Schlick part of the PBR-equation to take roughness into account.</p> <p><a href="https://learnopengl.com/PBR/IBL/Diffuse-irradiance">https://learnopengl.com/PBR/IBL/Diffuse-irradiance</a>, <a href="https://learnopengl.com/PBR/IBL/Specular-IBL">https://learnopengl.com/PBR/IBL/Specular-IBL</a>, <a href="https://learnopengl.com/Advanced-OpenGL/Cubemaps">https://learnopengl.com/Advanced-OpenGL/Cubemaps</a></p>
Shading	
Physically Based Shading	<p>Aside from a few (chosen) objects, everything in the rendered scene is shaded via the (Cook-Torrance BRDF) PBR shader. For this to work, we added special textures (albedo, aoRoughMetal, normal) with all the needed parameters encoded. The aoRoughMetal-texture holds the AO, roughness and metal values in the rgb channels in this order. In the vertex shader, everything is calculated to be in the same (world) space (for the <i>Normal Mapping</i> part, see next entry). The PBR shader supports an arbitrary amount of point and directional lights (array size set to 20, because this must be known at compiling) as well as IBL (see <i>Environment Map</i> entry). All these light sources are calculated differently and therefore are added onto each other, till it generates the final color. For point and directional lights, we only use the sample vector coming from this direction, to avoid integrating over the whole hemisphere. For IBL we have pre-computed maps (e.g. with importance sampling) to keep runtime calculations to a minimum.</p> <p>This PBR uses a distribution function for determining the amount of microfacets on the surface that are parallel to the halfway vector, which is affected by the roughness (e.g. smooth surface’s concentrated microfacets in one area). The two geometry functions are calculating how much a surface’s microfacets overshadow each other and</p>

	therefore reduce the light it reflects. With the Fresnel equation we determine how reflective a surface is from different angles. All calculations combined are giving an approximated physical shading.
	<a href="https://learnopengl.com/PBR/Theory">https://learnopengl.com/PBR/Theory</a> , <a href="https://learnopengl.com/PBR/Lighting">https://learnopengl.com/PBR/Lighting</a>
Simple Normal Mapping (not grading relevant)	Normal maps are used solely in our PBR shader. To bring everything into world space, we calculate the TBN-matrix in the vertex shader and pass it to the fragment shader, where we multiply it with the map's normal values and normalize those. After this, we can use them for light calculations.
	<a href="https://learnopengl.com/Advanced-Lighting/Normal-Mapping">https://learnopengl.com/Advanced-Lighting/Normal-Mapping</a>
<b>Post Processing</b>	
Bloom/Glow	This effect is applied on the lava and all PBR-shaded objects after they are drawn to a buffer (not the screen). In the corresponding fragment shaders, the "bloomColor" (every fragment with a color higher than a threshold) is outputted to a second, additional framebuffer-texture. This is blurred with another shader-program processing the buffer content with a gaussian blur using a ping-pong-method where the program is executed multiple times. Afterwards the blurred image is put onto the other and rendered on a single quad on the screen.  <i>Note:</i> The threshold in the lava shader is way smaller compared to the PBR one, because we wanted the lava to look like it is radiating heat. No PBR object is actually glowing, we just wanted to add the possibility.
	<a href="https://learnopengl.com/Advanced-Lighting/Bloom">https://learnopengl.com/Advanced-Lighting/Bloom</a> , <a href="https://learnopengl.com/Advanced-Lighting/HDR">https://learnopengl.com/Advanced-Lighting/HDR</a>

Additional Features	
<b>Additional Effects</b>	
Shell-Shading (Fur-Shading)	Shell-shading is applied only on Fluffy without any light calculations. A geometry-shader calculates multiple layers (shells) of the base object. The max. number of shells is restricted by hardware. Gravity and velocity are added on the shell position, according to the objects movements in the 3D room. In the fragment-shader hairs are drawn as points (given by a greyscale texture) on each shell.
	OpenGL Programming Guide, Eighth Edition, Shreiner David, Sellers Graham, Kessenich John, Licea-Kane Bill, 2013, S.510-532; <a href="http://www.xbdev.net/directx3dx/specialX/Fur/index.php">http://www.xbdev.net/directx3dx/specialX/Fur/index.php</a> , <a href="http://www.catalinzima.com/xna/tutorials/fur-rendering/">http://www.catalinzima.com/xna/tutorials/fur-rendering/</a>
<b>Sound</b>	
Sound-library	There is background music/lava, a sound for particles, a "happy sound" for throwing things into the lava, a "dying sound", "win and lose" sounds and a tutorial. The sound engine is mutable (see <i>List of Inputs</i> ).
<b>Code/Implementation</b>	
Pair Programming	Nearly everything was done in pair-programming either via discord or later on in person.
Singleton	GlobalManager is a singleton that holds all the level, window, etc. information that is accessible throughout the project.
Anti-Aliasing	We used multi-sampled buffers to decrease aliasing-artefacts. They are used before the bloom-effect, because they cannot be used for post-processing. Therefore a swap (blit) is executed to move the buffer content after the rendering to the multi-sampled buffer is completed.
	<a href="https://learnopengl.com/Advanced-OpenGL/Anti-Aliasing">https://learnopengl.com/Advanced-OpenGL/Anti-Aliasing</a>

List of Inputs	
WASD	Move Fluffy (3 <sup>rd</sup> person)
Mouse/Scroll	Rotate camera/zoom in and out
Space	Jump/Higer jump when held for a longer time
Enter	Advance from one state of the game to another at game-start-up or won/lost
ESC	Quit
F3/F4	Increase/decrease illumination multiplier
F1	Toggle wire-frame mode (will deactivate bloom)
F2	Toggle back-face culling
H	Toggle HUD
L	Increase score (always plus 11 points, makes level skippable)
N	Skip level
M	Mute and unmute sounds
P	Spawn 100 particles at (0.0f, 0.0f, 0.0f) with a 0.5f radius
Konami Code	Spawns random particles all over the lava (will most likely drop fps below 60 – this is an Easter egg – please do not take this into account while grading)

List of Libraries and Header-Files	
Object Loader	OBJ-Loader: <a href="https://github.com/Bly7/OBJ-Loader/tree/master/Source">https://github.com/Bly7/OBJ-Loader/tree/master/Source</a>
Physics	PhysX-Library 4.1: <a href="https://github.com/NVIDIAGameWorks/PhysX">https://github.com/NVIDIAGameWorks/PhysX</a>
GLM to PhysX converter	Converter: <a href="https://github.com/Caspila/GUInity/blob/master/Source/Converter.cpp">https://github.com/Caspila/GUInity/blob/master/Source/Converter.cpp</a>
Sound-Library	Irrklang 1.6: <a href="https://www.ambiera.com/irrklang/">https://www.ambiera.com/irrklang/</a>
HDR-Loader	stb_image 2.25: <a href="http://nothings.org/stb">http://nothings.org/stb</a>

Additional Sources	
Models	from Piotr Dyderski <a href="https://awesomepjt.org/">https://awesomepjt.org/</a> and Jakob Lindner <a href="https://jakob-lindner.com">https://jakob-lindner.com</a> and <a href="https://free3d.com/3d-model/rubber-duck-v1--614347.html">https://free3d.com/3d-model/rubber-duck-v1--614347.html</a> and <a href="https://sketchfab.com/3d-models/drums-d7beee9e9ed348ebae7e3b9fa09b7136">https://sketchfab.com/3d-models/drums-d7beee9e9ed348ebae7e3b9fa09b7136</a> and Stanford University Computer Graphics Laboratory and University of Utah
Sound effects	from <a href="https://www.zapsplat.com">https://www.zapsplat.com</a> and <a href="https://bigsoundbank.com/detail-0477-wilhelm-scream.html">https://bigsoundbank.com/detail-0477-wilhelm-scream.html</a>
Music	from Alex F. Römic <a href="https://soundcloud.com/spin360">https://soundcloud.com/spin360</a>
Textures	from <a href="https://share.substance3d.com/libraries/3866">https://share.substance3d.com/libraries/3866</a> and <a href="https://share.substance3d.com/libraries/5166">https://share.substance3d.com/libraries/5166</a> and <a href="http://www.hdrlabs.com/sibl/archive.html">http://www.hdrlabs.com/sibl/archive.html</a>