Praktikum: Echtzeit Computergrafik

Assignment 11 – *Explosions*

10 Points

In this assignment, there will be explosions! We'll extend our sprite renderer to be able to handle animated sprites to display pre-rendered explosions when enemies die.

Animated Sprites Preparation (3P)

For really nice particle effects like explosions we need to support animated sequences of sprites like e.g. the one in `external/art/05-Sprites/simple/explosion_b`. We will implement this by representing each sprite with a 2D texture array instead of a single 2D texture. Non-animated sprites are represented with arrays of size one.

- For each sprite texture, create a Texture2DArray instead of a Texture2D. Check the slides for additional information on how to create a Texture2DArray.
- Add the project "texAssemble" (from subdirectory "projects\DirectXTex") to your solution. Then, let your "ResourceGenerator" depend on that project.
- Modify your pixel shader to sample from a `Texture2DArray` instead of `Texture2D`. For now, just use 0 as the array slice index.

Checkpoint: Make sure your sprites still render!

- Extend the SpriteVertex definition by two float parameters $t, \alpha \in [0; 1]$. The animation parameter t describes the animation progress of a sprite and the scalar opacity factor α can be used to fade sprites in and out.
- Modify the sprite pixel shader such that t is used to sample the "array dimension" of the 2D texture arrays and that the alpha channel is multiplied with α. Note that the Sample-method expects the array dimension to be an integer (see slides!).

Checkpoint: Make sure you can still render your old sprites with the modified SpriteVertex and pixel shader. All sprites should now be sampled from array index 0, as we did not add any animations yet.

Adding basic explosions (4P)

Whenever an enemy dies, a simple explosion should be visible at its last position.

- Create the necessary resource .dds files for at least one sprite animation in your ResourceGenerator. The whole animation should be contained in a single .dds file.
    - Hint: Add the following lines to your NMake command line of your "ResourceGenerator" to convert a series of png images into a single dds file. This expects an input directory and an output file as parameter. (Note that these are

two single lines.):

```
call "$(SolutionDir)..\..\external\Tools\bin\dirassembler.bat"
"$(OutDir)texassemble.exe" "[input_path]"
"$(OutDir)resources\[output].dds"

"$(OutDir)texconv" -o "$(OutDir)resources" -f BC3_UNORM_SRGB
"$(OutDir)resources\[output].dds"
```

- Background Info: We will use the tool texAssemble for creating 2D texture arrays. Unfortunately, this tool is not capable of reading all the files from a given folder or specified by wildcard notation. Instead every filename must be given individually. Therefore we prepared the script dirassembler.bat which reads all filenames from a given directory, concatenates them to a single string and passes it on to the texAssemble tool. To finally compress the texture and to generate mipmaps we execute the texconv tool on the output file of texAssemble.

- Extend your `game.cfg` to define the appearance of an enemy explosion – the most basic definition must include the index of your sprite texture to be used and the duration of the explosion. Parse and store this configuration in your game initialization. Also add at least one of the explosion sprites from the art directory to your sprite list, of course.
- Create a struct that defines the dynamic state of an explosion. You'll basically need a position, a sprite texture index and the time that has passed since the explosion started, ranging from 0 (the animation just started) to 1 (the animation has finished).
- On enemy death, add a new explosion to a global list.
- Each frame, update the animation of all existing explosions depending on the time that has passed since the last frame. Remember to scale this time from 0 to 1 depending on the total duration of the explosion! Also make sure to delete explosions from your list when their animation is finished.
- On rendering, render each of the explosions. This works just like the already existing rendering of your projectiles: create a list of SpriteVertex into which you add all of your current explosions (don't forget to add the t parameter!), sort it and pass it to your sprite renderer. Make sure that you add all of your sprites (including those of the projectiles!) into a single list before sorting, otherwise your individual sprites will not be blended correctly.
  Check the slides for hints on how to handle those different sprites in a single draw call.

Checkpoint: You should now see an explosion on enemy death. Check if all explosions are correctly updated and also deleted when their animation is complete.

Advanced Particles (3P)

To make the explosions visually more appealing, we'll add a particle system to each of them: On enemy death, a number of 2D sprites flying in random directions will be emitted from the center of the explosion. This follows basically the same principle as your projectiles, so you can pretty much copy & paste a good amount of code.

Some words on artwork and configuration: For your explosion particles, select one of the explosions in `external/art/05-Sprites/simple/explosion_*`. It's up to you whether you use a complete animation for the particles' lifetime, a part of the animation or even just a

single frame (remember, you can also use the alpha-parameter of your particles to make them fade out!). Select whatever looks good to you.

We've left the "add everything to your configuration" part out of this exercise on purpose. Nevertheless – for your own sake, add everything you can think of to your configuration, like the number of particles spawned, the particle lifetime, min and max particle speed, texture, size, gravity influence and so on. If you want to make this look good, you'll definitely spend some time tweaking those parameters.

- Create a struct to store the dynamic state of an explosion particle storing position and direction as well as the lifetime and the sprite texture index. Again, scale the lifetime from 0 to 1 so you'll be able to also use animated sprites for the explosion particles.
- On enemy death, add a number of explosion particles to a global container. Initialize these particles at the center of the explosion and with a random direction (see slides). Choose the size as you like.
- Each frame, update the position and speed of all your explosion particles. This works just like the update for your guns' projectiles. In addition, advance the lifetime of each particle based on the elapsed time and a fixed maximum lifetime.
- On rendering, sort and pass all your explosion particles to your sprite renderer just like you did for the projectiles and explosions. Again, only use a single list for all of your sprites (explosions, projectiles, explosion particles).

Checkpoint: In addition to the basic explosion, you should see a number of particles hurled into random directions. Check if all those particles are correctly updated and also deleted when their animation is complete / their lifetime has exceeded the maximum lifetime.