Department of Computer Science

Augmented Reality 2018

13-02-2018

Group assignment 2 A

Deadline: **27-02-2018**

In this part of assignment 2, we will look into the computer graphics pipeline, in particular, generating meshes, creating shaders and achieving visual coherence. Note that this part of Assignment 2 is bigger than part B is going to be.

Mesh generation

Tasks

a) **1pt** Create a spaceship of any shape by building a mesh from scratch. The spaceship has to consist of at least 12 different triangles.

How many triangles does your ship have?

b) **1pt** Create a simple unlit shader that renders the spaceship, including vertex color.

How does your shader work?

c) **1pt** Animate the mesh of your space ship over time: Make some of the vertices move over time (e.g., the wings). Make parts of the spaceship blink by changing the vertices colors.

d) **2pt** Generate a terrain mesh between two image targets. Create a *terrain_base.jpg* image target and make it the **ARCamera's World Center**. Make sure it does not have any translation or rotation. Create a *terrain_controller.jpg* image target. Generate your terrain so it spans from $(0, 0)$ to the (x, z) position of the controller. Create a member to adjust the patch size of the terrain, i.e., the distance between rows and columns.
1pt Read the y position of each vertex from a height map.
1pt Make the height adjustable by using the orientation of the controller (e.g., the yaw of the controller's rotation).
1pt Set the uv coordinates so that you can apply a surface texture to your terrain.

Min / Total = **3.0pt / 8.0pt**

TIPS

- Access the Mesh class to generate your meshes.
- You can use `mesh.RecalculateNormals()` to easily set the normals of a mesh after setting vertices and indices. Be careful with reusing vertices (i.e., indexing the same vertices multiple times). Sharing normals between triangles is not always desired. You can create vertices with the same position, but different normals to create sharp edges.
- Note that the number of vertices and indices largely varies depending on the patch size and the position of the controller. Clear your mesh before updating it every frame.
- Note that since your terrain base is the world center, you do not have to think about local and global coordinates when generating the terrain mesh.
- You can choose your approach for how you generate your terrain mesh. There are different sources and tutorials for generating terrain meshes from height maps. For instance, adamdawes.com has a good conceptual introduction. jayelinda.com has a long tutorial in CSharp, whereas we only need parts of it.

Visual coherence

PREPARATION

- Download OpenCV and MatDisplay from the Exercises page on Blackboard.
- Uncheck *Enable video background* in the *Vuforia configuration*.
- Set the aspect ratio of your *Game view* to the aspect ratio of your webcam (should be 4:3). Leaving it at "Free aspect" is a common cause for introducing visual offsets. You can also add a configuration with the resolution of your webcam (normally 640x480).
- Draw the camera image background using *MatDisplay*.
- Set the field-of-view of your virtual camera to 41.5 (standard value for webcams). You have to do this in your script, since (for unknown reasons) Vuforia sets the camera FoV to 60 during startup if the video background is disabled.
- If everything is done correctly, it should initially look the same as with Vuforia's video background and augmentation should still work. Now we have the freedom to display any post-processed video background.
- Note: you might need to activate Vuforia's video background again when switching to tasks of the previous section or draw the video background in these tasks as well.

TASKS

a) **1pt** Implement the phantom technique (using `ColorMask`) to let an image target correctly occlude virtual objects.

1pt Allow for any 2D shape, i.e., cutouts to occlude virtual objects correctly. The occlusion mask can be given as a monochrome texture (e.g., the beer mask in the templates on Blackboard). Use `discard` in your shader to omit pixels.

1pt Attach *ruediger_rigged.fbx* (contained in the 3D models on Blackboard) to an image target. Rotate the head so that the penguin always looks at the beer image target (the head GameObject is a child of the rigged model). You can use `Quaternion.LookRotation` (use the up-vector of the penguin's spine for better results). Hint: For a correct rotation you have to save the initial rotation of the head during startup to concatenate with your calculated rotation at runtime.

b) **1pt** Display the camera image in grayscale and create a shader that renders the augmentation in grayscale as well.

2pt Add noise to the rendering to resemble the noise of the (grayscale) video feed.

c) **1pt** Draw the contours of the camera video feed and create a shader that renders the contour of the virtual object in black and the inner part in white. The `Imgproc.adaptiveThreshold` OpenCV function can be used to quickly generate a simple stylization (please note that this is generally not the purpose of adaptive thresholding).

1pt Add monochrome noise to your shader so that it resembles the noise of the video feed after stylization.

Min / Total = **3.0pt / 8.0pt**