CG1 WS 23/24 - Exercise 4: Texturing

Technische Universität Berlin - Computer Graphics
Date 14. December 2023 **Deadline** 17. January 2024
Prof. Dr. Marc Alexa, Tobias Djuren, Hendrik Meyer, Markus Worchel

Texture Mapping (6 points)

This exercise is about mapping a texture onto an object using WebGL and Three.js. The basic setup contains two windows where a texture image is shown in the left subwindow and a textured object in the right subwindow. Several texture images are provided in the archive file. The tasks in detail are:

1. (Basics) Divide the application into two windows with equal size and add the gui (`Settings` and `createGUI` are provided in `helper.ts`). On the right side setup an empty scene using `setupCamera` and `setupControls` in `helper.ts`. On the left show the texture image that is selected in the *Texture* selector.

   We provide the `ImageWidget` class in `imageWidget.ts` to display images. Like the `RenderWidget` you can pass a window instance to the constructor to add the widget to a window. You can then set the image given a path with `setImage`. *(0.25 points)*

2. (UV Attribute Shader): Write a simple (vertex and fragment) uv shader, that colors every pixel of the object with the selected texture using the interpolated uv attribute provided by `three.js`. Test your implementation with the Box, Sphere and Knot geometries provided by `createBox`, `createSphere` and `createKnot` in `helper.ts`. The three models should be switchable using the *Geometry* selector of the gui. The shader should be applied when *UV attribute* is the selected *Shader*. *(1 points)*

   Hint: You have to pass the texture as uniform to make it accessible in the shader. The `GLSL ES` counterpart is `sampler2d`. You can interpolate the texture using `texture2D` (or `texture` in `GLSL ES` version 3.00).

3. (Quad): *Geometry and uv coordinates are not always given:* Create a quad using `BufferGeometry` and two triangles. The quad should have size 2 by 2. You have to manually add the position and uv attributes so that the full texture is visible on the quad. The quad should also be switchable using the *Geometry* selector of the gui. *(1.0 points)*

4. (Drawind Pen): Extend your uv shader so that you are able to draw with a white pen on the canvas. The drawing should be clearable using the *Clear Drawing* gui button. *(0.5 points)*

   Hint: `ImageWidget` already provides the drawing functionality (see `enableDrawing` and `clearDrawing`), you just need to convert the `ImageWidget.DrawingCanvas` to a texture (see `CanvasTexture` of `Three.js`) and update the texture uniform every time `ImageWidget.DrawingCanvas` triggers the `"updated"` event.

5. (Spherical Mapping): *In case you have no predefined uv coordinates an universal mapping is needed:* Add the bunny to the selectable geometries (see `createBunny` in `helper.ts`). The bunny has no predefined uv attributes, therefore the *UV attribute* shader can not be used. Calculate uv coordinates for every vertex of the current selected model using a *spherical mapping*. The result should be visible when *Spherical* is selected in the *Shader* selector. *(0.75 points)*

   Hint: Although you can calculate the uv coordinates offline and pass them to the shader using attributes, you can also calculate them in the vertex shader. We recommend to calculate them in the vertex shader as this makes the following tasks much easier.

6. (Fix Spherical Mapping) Simple spherical mapping leads to artifacts when vertices in a triangle are mapped to different sides of the texture. Create new shaders that avoid such artifacts. The result should be visible when *Spherical (fixed)* is selected in the *Shader* selector. *(0.5 points)*

   Hint: Think about where the uv coordiantes should be calculated.

7. (Environment Mapping) Implement environment mapping shader where the selected texture represents the environment. The selected geometry should be shaded as if it would reflect the environment as shown in the exemplar solution. The checkbox *Enviroment* should toggle the background between black and the selected texture. *(0.75 points)*

   Hint: You can set the background of the scene to a color or to a texture in `Three.js`. For environment maps as background you need to set the `mapping` property of the `Texture` to the correct mapping mode (`EquirectangularReflectionMapping`).

8. (Normal Mapping): The following sub task build upon each other and should be implemented in one shader (pair). The shader should be applied when *Normal Map* is the selected shader.

   - (Adding Light) Implement a shader that calculates an ambient and diffuse illumination model like in the last exercise, you can add a specular component as well. You are free to choose and hard code the parameters: light position, magnitude, reflectance and specular light. The ambient color (ambient light intensity) and diffuse color (diffuse light intensity) should be the texture. The uv attribute should be used for the uv coordiantes as in task 2. *(0.5 points)*

     Remark 1: As the bunny has no uv attribute this shader does not need to work for the bunny. *Remark 2:* If you proceed to implement the normal map, the light calculation only needs to work on the quad.

   - (Normal Mapping): *This only needs to work for the quad.* Extend the shader that it can handle normal maps. The different normal maps should be selectable using the *Normal Map* selector in the gui. You need know the tangent, (bi)tangent and normal direction for the vertices of the quad geometry so you are able to tweak the normals for the shading calculation. *(0.75 points)*

     Hint: First: The normals of the normal maps stored in the image are defined in tangent space scaled by $\frac{1}{2}$ with an offset of 0.5 so that they fit in $[0, 1]^3$. The basis of this space is the tangent in u direction (red), the (bi)tangent in v direction (green) and normal(blue). Second: If you oriented your quad as one would expect on the xy-plane, the normal, tangent and (bi)tangent vectors are constant over the whole quad and equal to the world coordinate axes. This significantly simplifies the computation. You do not need to implement the more complicated case on general geometry.

Requirements

- Exercises must be completed individually. Plagiarism will lead to exclusion from the course.
- Submit a `.zip` file of the `src` folder of your solution through ISIS by **17. January 2024, 23:59**.
- *Naming convention*: {firstname}_{lastname}_cg1_ex{#}.zip (for example: jane_doe_cg1_ex4.zip).
- You only hand in your `src` folder, make sure your code works with the rest of the provided skeleton.