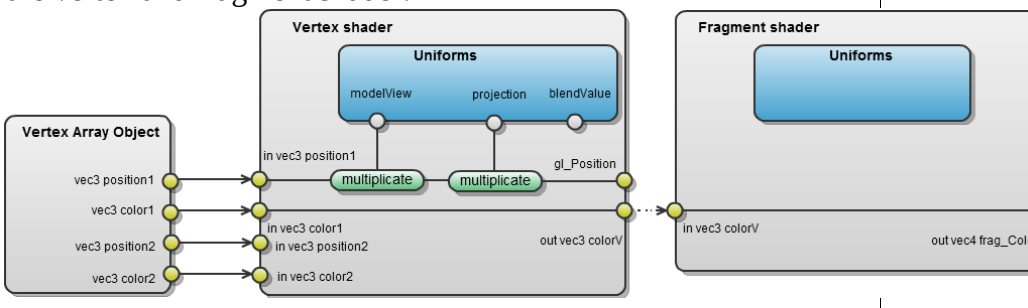
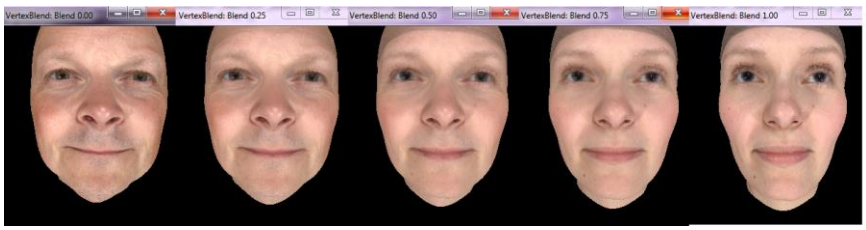
## *Exercise 02561-03: Introduction to shaders and GLSL*

| | |
|---|---|
| Reading | 4.1 - 4.4.2 |
| Purpose | In this exercise you will learn how to setup both shader uniforms and vertex attributes in both shader code and C++ code.<br>Also you will learn how vertices are transformed through the different stages at the pipeline.<br>The exercise will also introduce you to a very simple type of animation called vertex blending (also known as morphing) performed in the vertex shader using GLSL. |
| Setup | The program 02561-03-01 loads two face model and displays the result of one. The user can rotate the camera around the model by left mouse drag.<br>The model data is loaded from a file format called VTK (used at DTU-IMM) by a model loader defined in the file "vtk_reader.h".<br>The data is transformed into the vertex format (position and color), uploaded to the vertex buffer and mapped to the vertex array object.<br>The following image shows how the vertex stream flows through the vertex and fragment shader:<br>  <br><br>The data in the provided files has the same number of vertices and there is topological correspondence between vertices of different file set (this means the vertices defining a part of the face has the same indices in all files).<br>The program already has a `blendValue` which value can be changed with mouse wheel or '+' / '-' key. The `blendValue` is showed in the title bar. The `blendValue` is mapped to the uniform of same name.<br><br>Hint: Programming shaders can be tricky, since you cannot use the C++ debugger to set breakpoints in the shader code. To debug shader code, you would instead in the shader change a value to a color and write out that color in the fragment shader. |

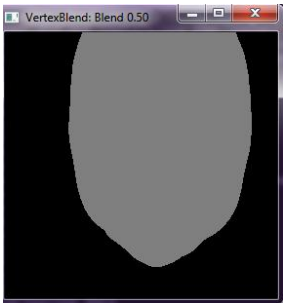
## *Exercise 02561-03: Introduction to shaders and GLSL*

Part 1
Vertex blend in GLSL

In this exercise we would like to blend between the two faces using the vertex shader. The final result can be seen in the image below:




- Modify the vertex shader (`blending.vert`) to use position2 and color2 instead of position1 and color1. Also try to use mixed combinations (position1 with color2).
- To make sure that the blendValue is correctly setup in the vertex shader, assign blendValue to colorV. To do this you need to create a vec3 by calling one of the vec3 constructor functions `vec3(float)` or `vec3(float, float, float)`. When changing the blendValue in the running program (using '+'/'-' keys) you should see a gray outline. Warning: the image may be black before you change the blendValue.



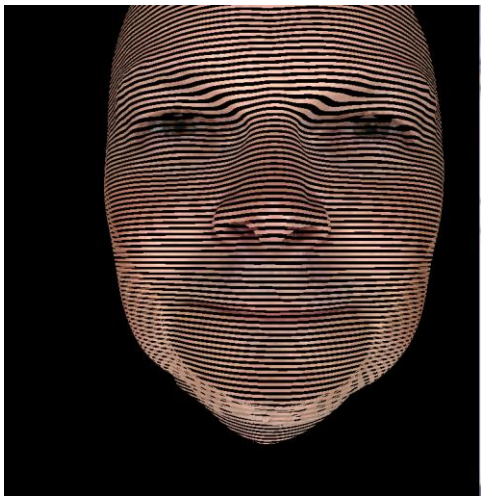

- Modify the vertex shader to blend between the two positions and two colors based on the blend value. Use the following formula for this:
  $m = (1 - b)p_1 + bp_2$
  , where b is the blend value.

You should now be able to blend between the two faces.

## *Exercise 02561-03: Introduction to shaders and GLSL*

| | |
|---|---|
| Part 2<br>Normal extrusion | The VTK file-format also contains vertex normals, which is currently not used. Vertex normals are usually used for computing lighting, but can also be used for other effects such as normal extrusion. In normal extrusion the vertex position is moved in the direction to the normal. This gives some funny looking faces:<br>  <br>• Add a `normalExtrusion` float variable in the C++ code and let keys 'n' and 'm' control the value. Modify the `updateTitle()` function to use both blend value and normal extrusion value.<br>• Add `normalExtrusion` as as vertex shader uniform and assign the value in the shader in the `display()` function. (You can test that your setup is correct using the technique from part 1)<br>• Extend the `Vertex` struct to contain two normals as well.<br>• Extend the vertex shader to take the two normals as vertex attributes. Modify the C++ program so the vertex array object is configured correctly. (Again it is a good idea to test to see that the setup is correct).<br>• Modify the vertex shader to move the vertex along the normal direction by the amount specified by normalExtrusion. This should be done before transforming the position into clip-space.<br>You should now be able to use both normal extrusion and vertex blending. |
| Part 3<br>Questions | Answer the following questions:<br>• What is the difference between a vertex attribute and a vertex uniform variable used in a shader? When can they be changed?<br>• Explain in general terms how a vertex shader works.<br>• Explain in general terms how a fragment shader works.<br>• Describe what happens between the vertex shader and the fragment shader.<br>• Is it possible to blend the color in the fragment shader instead of the vertex shader? If so, it it a good idea doing this? |

## *Exercise 02561-03: Introduction to shaders and GLSL*

| | |
|---|---|
| Part 4<br>Optional<br>Discarding fragments | Optional<br>In the fragment shader, it is possible to discard a fragment which means that no value is written to the color buffer or the z-buffer.<br>• Transfer the vertex position (in model coordinates) from the vertex shader to the fragment shader.<br>• Update the fragment shader to discard every pixel which has an even value for the model coordinate's y-value.<br>**Hint**: You can use the following GLSL functions:<br>http://www.opengl.org/sdk/docs/manglsl/xhtml/round.xml<br>http://www.opengl.org/sdk/docs/manglsl/xhtml/mod.xml<br>Discarding a pixels is done simply by calling `discard` followed by a semicolon – it is not a function:<br>`discard;`<br>  <br>The result of discarding the pixels should look like this: |