Tehtävä 3. 3D-meshin luonti ja näyttäminen ruudulla.

Tehdään sovellus, joka luo teapot meshin ja näyttää sen ruudulla. Ensin tee uusi scene, ja lisää se projektiin ja laita käynnistymään se ohjelman alussa. Nimeksi voisi antaa vaikkapa SimpleMeshRenderingScene tms.

Fragment -shader koodi (nimeä esim. Simple3d.vs):

```
// default to medium precision
precision mediump float;

#ifdef GL_FRAGMENT_PRECISION_HIGH
   #define NEED_HIGHP highp
#else
   #define NEED_HIGHP mediump
#endif


void main()
{
    gl_FragColor.rgba  = vec4(1.0f, 0.2f, 0.5f, 1.0f);
}
```

Vertex shader koodi (nimi esim Simple3d.fs):

```
uniform mat4    g_matModelViewProj;

attribute vec4  g_vPositionOS;
attribute vec3  g_vNormalOS;

void main()
{
    vec4 vPositionCS = g_matModelViewProj * g_vPositionOS;

    // Pass everything off to the fragment shader
    gl_Position  = vPositionCS;
}
```

Tässä valmis Teapot -meshin luonti funktio (incluudaa teapot.h, joissa on data teekannulle):

```
graphics::Mesh* createTeapotMesh()
{
    //Create Index buffer
    graphics::IndexBuffer* ib = new graphics::IndexBuffer(TeapotData::indices, TeapotData::numIndices);

    //Create Vertex arrays
    graphics::VertexArray* va[] =
    {
        new graphics::VertexArrayImpl<slmath::vec3>(
            graphics::ATTRIB_POSITION, (slmath::vec3*)TeapotData::positions, TeapotData::numVertices),

        new graphics::VertexArrayImpl<slmath::vec3>(
            graphics::ATTRIB_NORMAL, (slmath::vec3*)TeapotData::normals, TeapotData::numVertices)
    };

    // Create vertex buffer from vertex arrays
    graphics::VertexBuffer* vb = new graphics::VertexBuffer(&va[0], sizeof(va) / sizeof(va[0]));

    // Create mesh from ib and vb
    return new graphics::Mesh(ib, vb);
}
```

Muokkaa SharedShaderValues ja GlobalShaderUniforms rakenteita niin, että total timen sijasta välitetään shaderille. Yksi 4x4 matriisi nimeltään modelViewProj (yhdistetty model-view-projection matriisi (voit toki jättää sen totalTimen sinne, siitä ei pitäisi olla haittaakaan):

```
struct SharedShaderValues : public core::Object
{
    slmath::mat4 matModelViewProj;      // Model view projection matrix. Used to transform position vertices to clip space.
};

class GlobalShaderUniforms : public graphics::ShaderUniforms
{
public:
    GlobalShaderUniforms(graphics::Shader* shader, const SharedShaderValues* shaderShaderValues = 0)
        : ShaderUniforms(shader)
        , m_shaderShaderValues(shaderShaderValues)
    {
    }

    virtual ~GlobalShaderUniforms()
    {
    }

    virtual void getUniformLocations(graphics::Shader* shader)
    {
        // Get uniform locations
        m_id = glGetUniformLocation(shader->getProgram(),   "g_matModelViewProj");
    }

    virtual void bind(graphics::Shader* shader)
    {
        shader->bind();
        if( m_shaderShaderValues )
        {
            if (m_id >= 0)
                glUniformMatrix4fv( m_id, 1, GL_FALSE, &m_shaderShaderValues->matModelViewProj[0][0]);
        }
    }
private:
    const SharedShaderValues*   m_shaderShaderValues;
    GLint                       m_id;
};
```

SimpleMeshRenderingScene -luokkaan pitää tehdä uusia jäsenmuuttujia matriiseille:

```
private:
    slmath::mat4 m_matProjection;
    slmath::mat4 m_matView;
    slmath::mat4 m_matModel;
```

SimpleMeshRenderingScene updatessa laske matriisien arvot käyttäen slmathin tarjoamia funktioita:

```
// Camera perspective matrix = field of view, aspect ratio, near plane distance and far plane distance
float fAspect = (float)esContext->width / (float)esContext->height;
m_matProjection = slmath::perspectiveFovRH(
    slmath::radians(45.0f),
    fAspect,
    5.0f,
    1000.0f);

// Look at view matrix = eya point, look at point and world up vector
m_matView = slmath::lookAtRH(
    slmath::vec3(0.0f, 70.0f, 70.0f),
    slmath::vec3(0.0f, 15.0f, 0.0f),
    slmath::vec3(0.0f, 1.0f, 0.0f));

// Update teapot model matrix
m_matModel = slmath::rotationX(-3.1415f*0.5f); // -90 degrees around x first
m_matModel = slmath::rotationY(m_totalTime) * m_matModel; // Rotate according to total time
m_matModel = slmath::translation(slmath::vec3(0.0f, 0.0f, 0.0f)) * m_matModel; // Translate
```

SimpleMeshRenderingScene alustuskoodia shaderin luonnin ja meshin luonnin osalta (Muista tottakai lisää core::Ref<graphics::Mesh> m_mesh; jäsenmuuttujaksi scenelle):

```
FRM_SHADER_ATTRIBUTE attributes[2] =
{
    { "g_vPositionOS", graphics::ATTRIB_POSITION },
    { "g_vNormalOS", graphics::ATTRIB_NORMAL }
};


// Load shader
core::Ref<graphics::Shader> shader =
        new graphics::Shader("assets/Simple3d.vs", "assets/Simple3d.fs",
        attributes, sizeof(attributes) / sizeof(FRM_SHADER_ATTRIBUTE));

// Create mesh
m_mesh = createTeapotMesh();
```

Renderöintikoodi asettaa viewportin, tyhjentää näytön, asettaa sopivan OpenGL piirtotilan, laskee model view projection matriisin, bindaa materiaalin ja rendaa meshin:

```
// Set the viewport
glViewport( 0, 0, esContext->width, esContext->height );

// Clear the backbuffer and depth-buffer
glClearColor( 0.0f, 0.8f, 0.8f, 1.0f );
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
checkOpenGL();

// Initialize GL state.
glDisable(GL_BLEND);
glEnable(GL_CULL_FACE);
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LEQUAL);
checkOpenGL();

// These values are needed to update for each mesh which is rendered (different model matrix)
slmath::mat4 matModelView           = m_matView * m_matModel;
slmath::mat4 matModelViewProj       = m_matProjection * matModelView;

// Set matrix to shared values
m_sharedValues->matModelViewProj    = matModelViewProj;

// Bind material (sets uniform values)
m_material->bind();
checkOpenGL();

// Render the mesh using active material.
m_mesh->render();
checkOpenGL();
```

Lopputuloksen pitäisi näyttää tältä: