Custom Shader for 2D Effects on 3D Devices

Purpose

2D Effects on 3D devices are realized by usage of shader in the RenderDevice2D for implementation of 2D over 3D.
Several times, available effects for such devices have been enhanced. Please refer to Open GL related Effects below for more information.

- OpenGL Solid Color Brush Mask Effect
- OpenGL Bitmap Brush Mask Effects
- OpenGL Gradient Brush Effects
- OpenGL Flip Effects
- OpenGL Drop Shadow Effects
- OpenGL Outline Effects

For more flexibility and the option, to write custom 2D effects with a custom shader, the API has been enhanced:

- It is possible to provide a custom shader source from an effect implementation
- It is also possible to transfer uniforms (definition and value) from an effect implementation

The API

The API is in class `Candera::Internal::Context2DOver3DDevicePool`:

```
/**
 *  Set a custom program.
 *  @param vertexShader  The vertex shader code
 *  @param pixelShader  The pixel shader code
 *  @param uniforms  An array with uniform definitions (@note count is limited)
 *  @param uniformCount  Number of uniform definitions
 *  @param guid  A guid to distinguish custom programs
 */
```

```
void SetCustomProgram(const void* vertexShader, const void* pixelShader, CustomUniform
uniforms[],
□□□□□  UInt8 uniformCount, FeatStd::Internal::Guid const& guid);

/**
 *  Reset usage of a custom program.
 */
void ResetCustomProgram();

/**
 *  Check if a custom program is set.
 *  @return  True, if a custom program is set. False, otherwise.
 */
bool HasCustomProgram() const;
```

> The API is internal and shall only be used by experienced users!

Furthermore, the RenderDevice2D API has been enhanced to set a second surface (texture).

```
/**
 * List of surface types.
 */
enum SurfaceType{
□DestinationSurface = 0,   ///< Destination surface.
□SourceSurface = 1,        ///< Source surface.
□MaskSurface = 2,          ///< Mask surface.
□SecondSourceSurface = 3   ///< Further Source surface.
};
```

GlMultiTextureBrushBlend

A new effect, `GlMultiTextureBrushBlend`, has been added with the purpose to demonstrate the usage of the new API.

> The appearance of this effect in Scene Composer can be viewed on page Open GL Multi Texture Brush Blend

This effect uses the following shader code in its implementation:

```
static const Char* s_vertexShader = {
    "#ifndef GL_ES\n"
    "#define lowp\n"
    "#define mediump\n"
    "#define highp\n"
    "#define precision\n"
    "#endif\n"
    "precision highp float;\n"

    "attribute vec4 a_Position;\n"
    "attribute vec4 a_TextureCoordinate;\n"

    "uniform mat4 u_Transform;\n"
    "uniform mat4 u_TexTransform;\n"

    "varying vec2 v_TexCoord;\n"

    "void main(void)                                                \n"
    "{\n"
    "   gl_Position = a_Position * u_Transform;                     \n"
    "   v_TexCoord.xy = (a_TextureCoordinate * u_TexTransform).xy; \n"
    "}\n"
};

static const Char* s_fragmentShader = {
    "#ifndef GL_ES\n"
    "#define lowp\n"
    "#define mediump\n"
    "#define highp\n"
    "#define precision\n"
    "#endif\n"
    "precision highp float;\n"

    "uniform sampler2D u_Texture;\n"
    "uniform sampler2D u_Texture2;\n"
    "uniform vec4 u_ConstTexColor;\n"
    "uniform float u_TexturesBlendFactor;\n"

    "varying vec2 v_TexCoord;\n"
```

```
    "void main(void)\n"
    "{
\n"
    "    vec4 color = u_ConstTexColor;
\n"
    "    vec4 texCol = texture2D(u_Texture, v_TexCoord) * u_TexturesBlendFactor;
\n"
    "    vec4 texCol2 = texture2D(u_Texture2, v_TexCoord) * (1.0F - u_TexturesBlendFactor);
\n"
    "    gl_FragColor = color * (texCol + texCol2);
\n"
    "}\n"
};
```

> The shader code here is written for OpenGL ES 2.0 on purpose! The sample effect shall run also on devices supporting only this version.

The core functionality inside the `GlMultiTextureBrushBlend::Render` method:

```
static_cast<void>(ActivateSecondaryImage(output));

Internal::Context2DOver3DDevicePool& pool =
Internal::Context2DOver3DDevicePool::GetInstance();
FeatStd::Internal::Guid guid("8D86313D-47B2-49BD-89C3-DBD24E23A5A1");

Float texturesBlendFactor = m_texturesBlendFactor.Get();
texturesBlendFactor = (texturesBlendFactor < 0.0F) ? 0.0F : texturesBlendFactor;
texturesBlendFactor = (texturesBlendFactor > 1.0F) ? 1.0F : texturesBlendFactor;

Internal::Context2DOver3DDevicePool::CustomUniform customUniforms[1] = {
{ Internal::Context2DOver3DDeviceProgram::FloatUniform, "u_TexturesBlendFactor",
texturesBlendFactor }
};

pool.SetCustomProgram(s_vertexShader, s_fragmentShader, customUniforms, 1, guid);

m_bitmapBrush.Render(input, inputArea, transform, node, output, outputArea);
if (pool.HasCustomProgram()) {
pool.ResetCustomProgram();
}
```

```
success = DeactivateSecondaryImage(output) && success;
```

As seen above, the value from the effect property `m_texturesBlendFactor` is normalized and then provided as uniform `u_TexturesBlendFactor`, which is used by shader code defined with `s_vertexShader` and `s_fragmentShader`.

> Parameters for the shader are provided as uniforms. For the user, the parameters are exposed as Effect properties!

Revision #15
Created 2024-06-11 12:18:35 UTC by Thomas Meikl
Updated 2026-09-25 05:21:23 UTC by Kanai Tomoaki