splatty!

The gaussian splatting viewer for everyone!

cppNorth!

The best developer conference for everyone!

meet splatty!

meet splatty!

meet splatty!

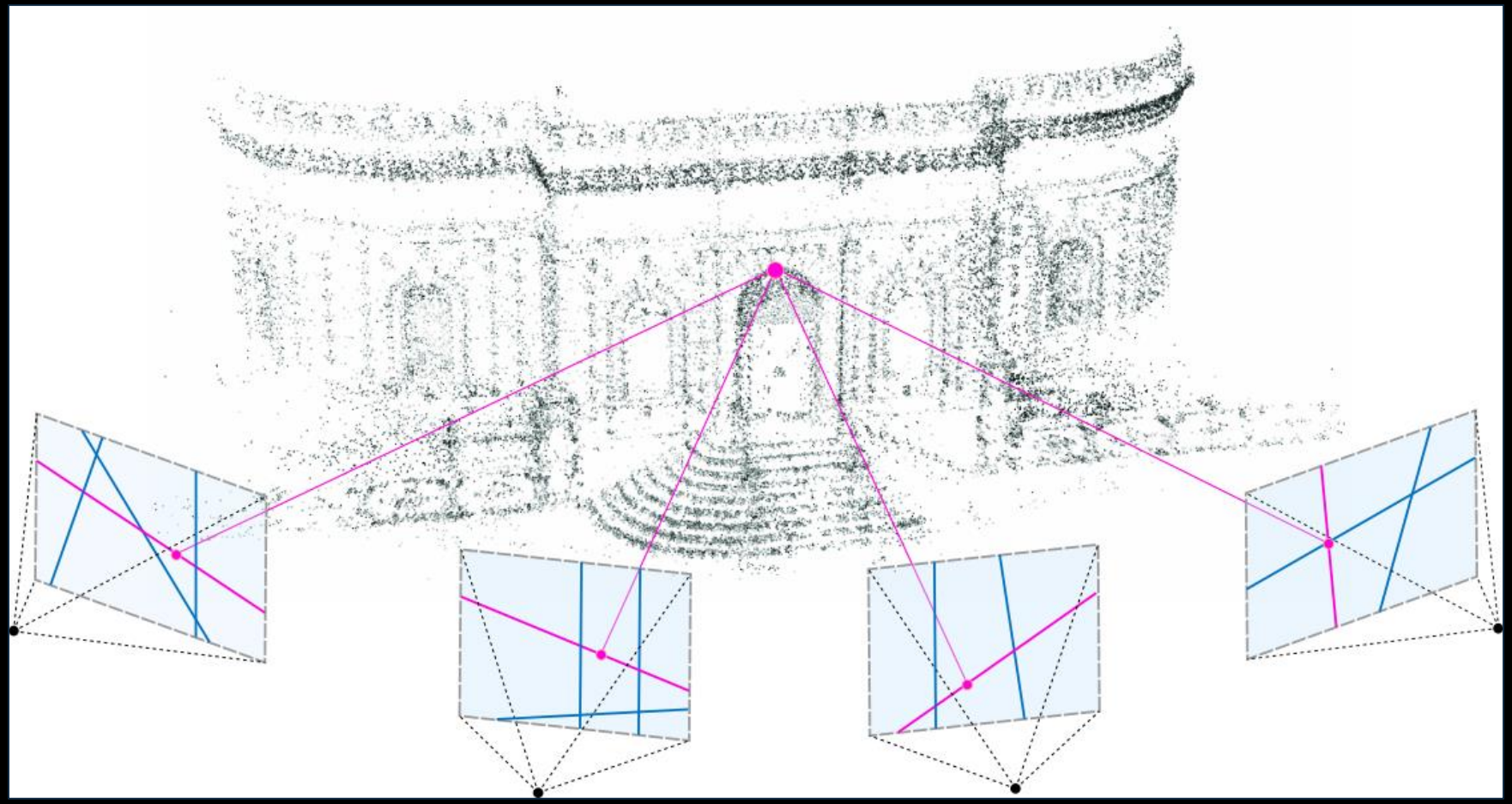
meet splatty!

cppNorth 2024

What is splatting grandpa?

The Splatting!

Structure from motion!



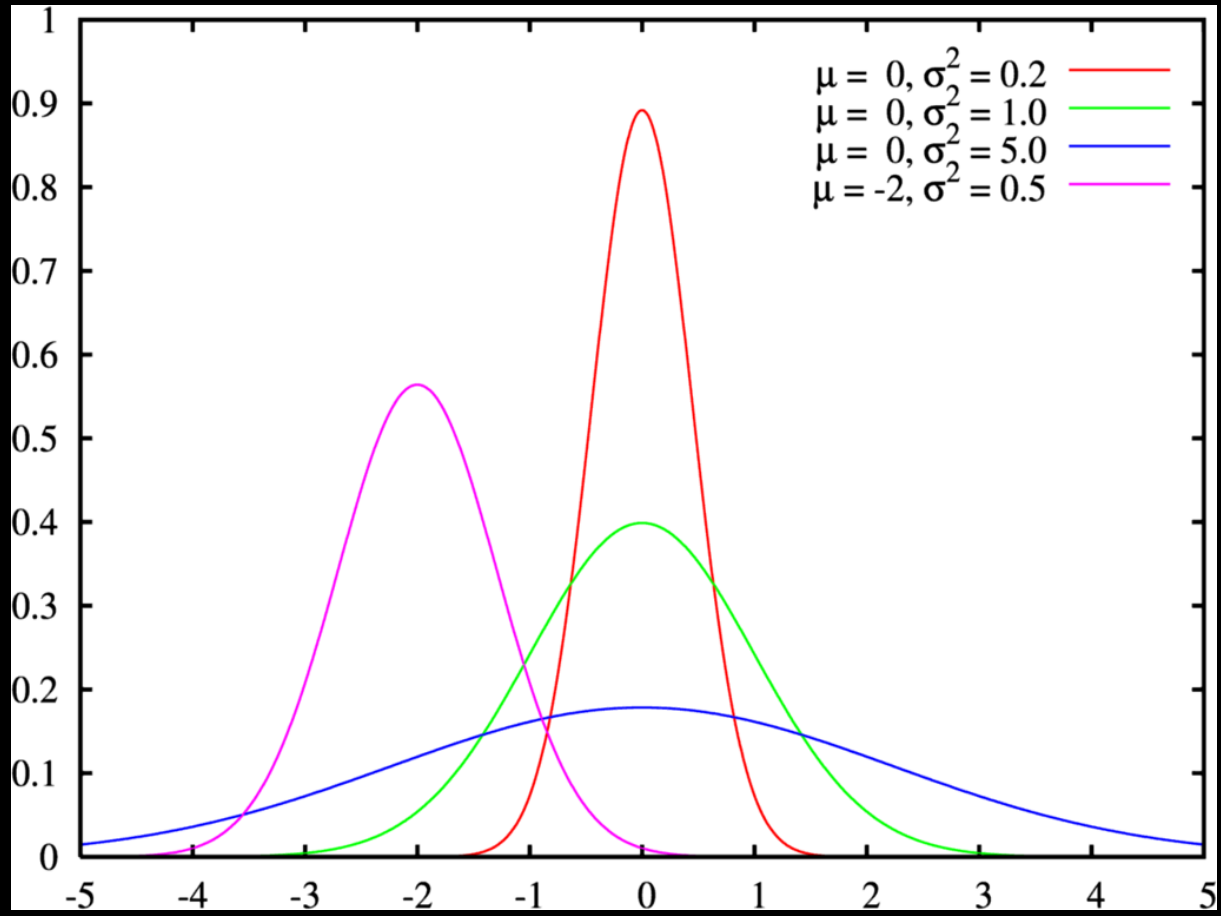
What is a gaoh-see-anne grandpa?

The Gaussian!

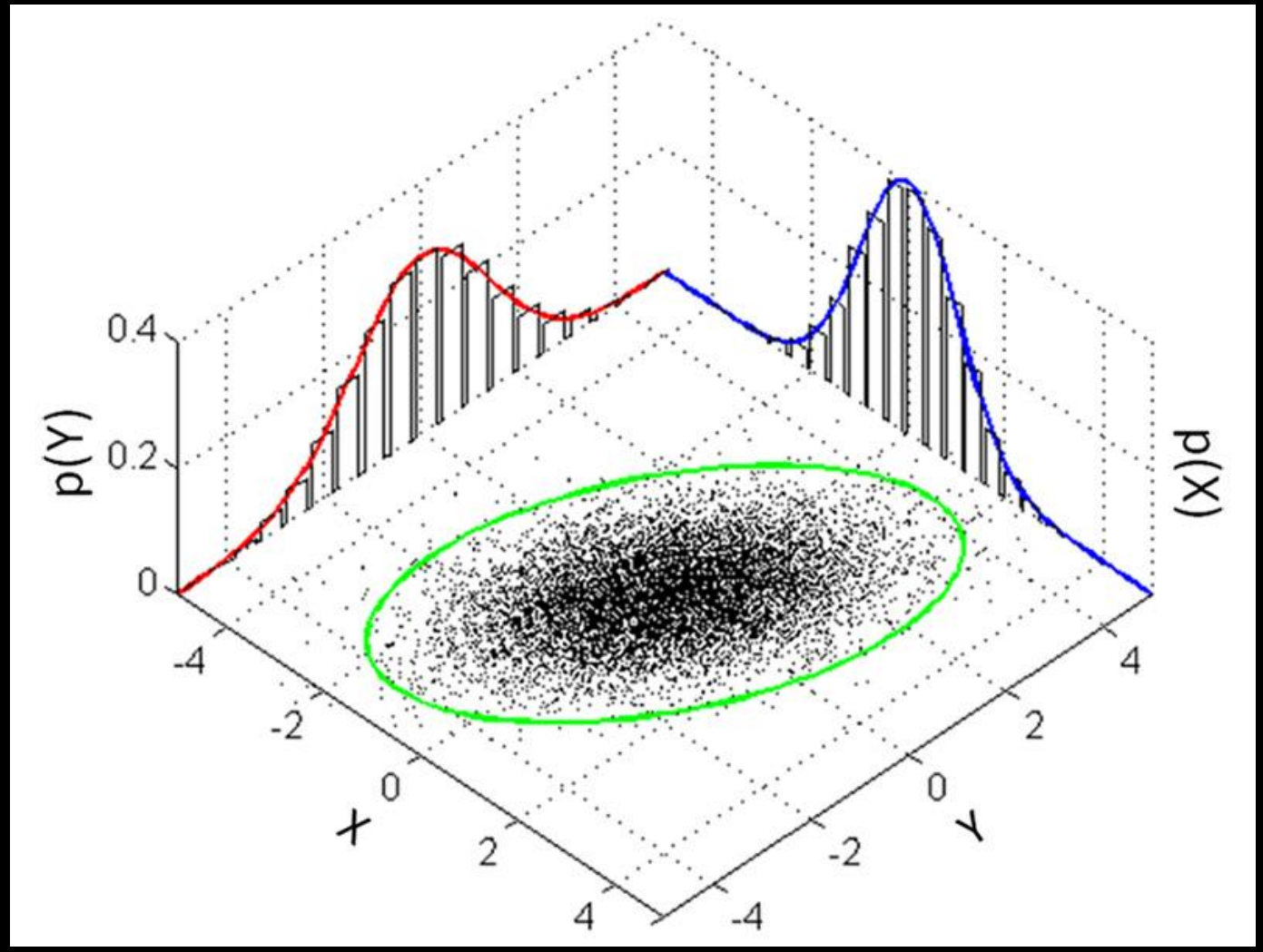
Carl Friedrich Gauß
1777–1855

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right)$$

The Gaussian!

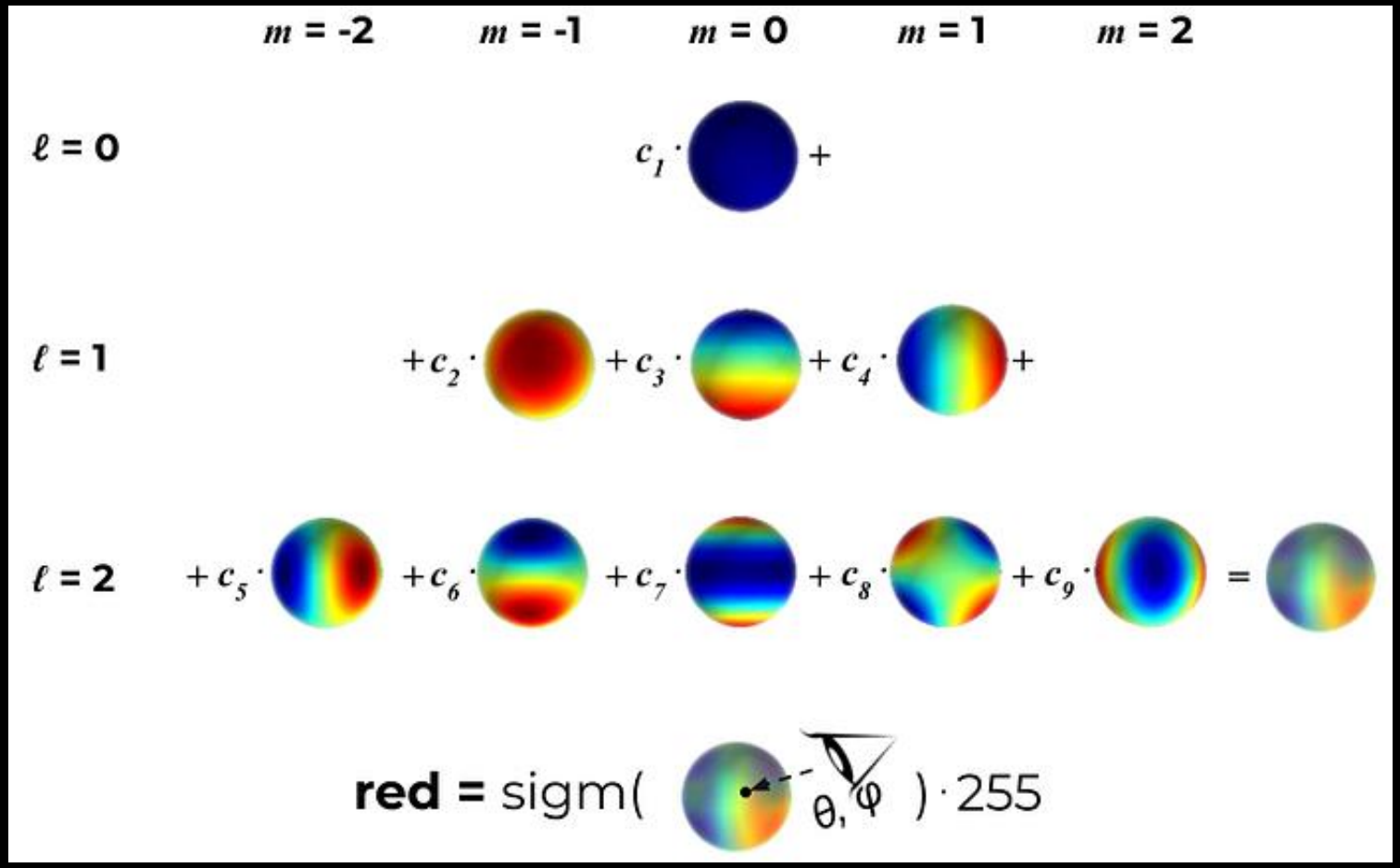


The multi-dimentional Gaussian!



Adaptive densification!

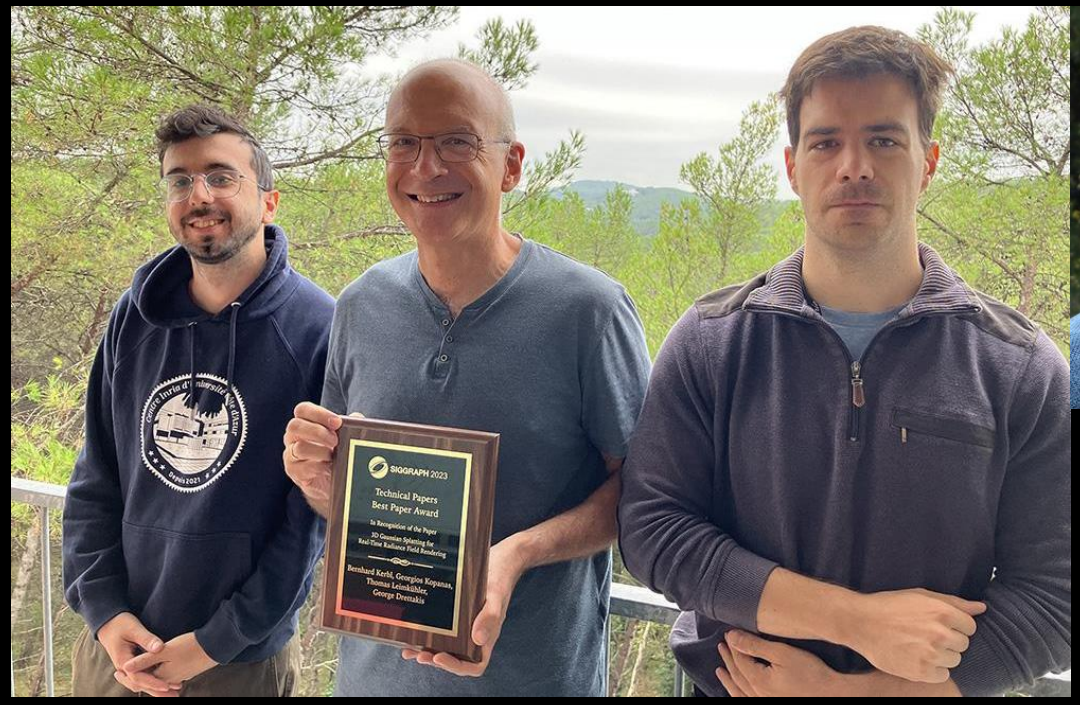
View dependent color!



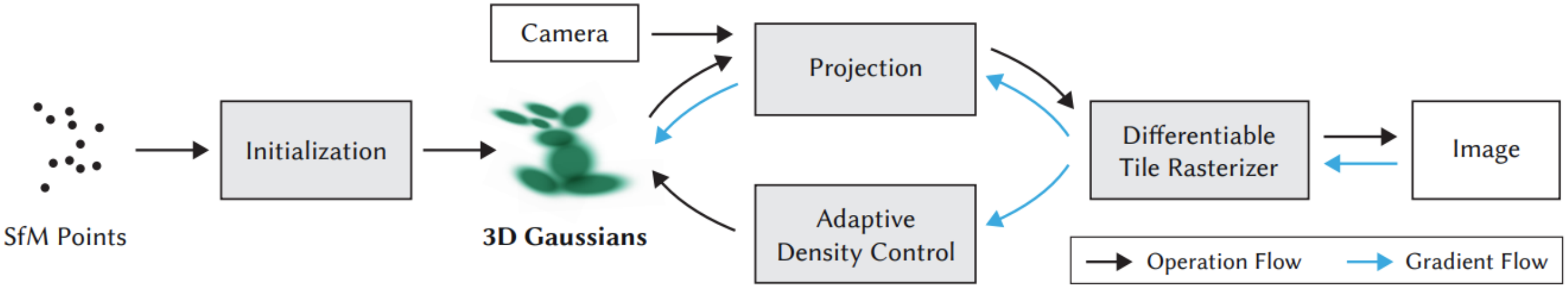
The 3D Gaussian Splatting!

3D Gaussian Splatting for Real-Time Radiance Field Rendering

BERNHARD KERBL*, Inria, Université Côte d'Azur, France
GEORGIOS KOPANAS*, Inria, Université Côte d'Azur, France
THOMAS LEIMKÜHLER, Max-Planck-Institut für Informatik, Germany
GEORGE DRETTAKIS, Inria, Université Côte d'Azur, France



The 3D Gaussian Splatting!



See grand-pa, it’s easy!

Let’s define today’s goals!

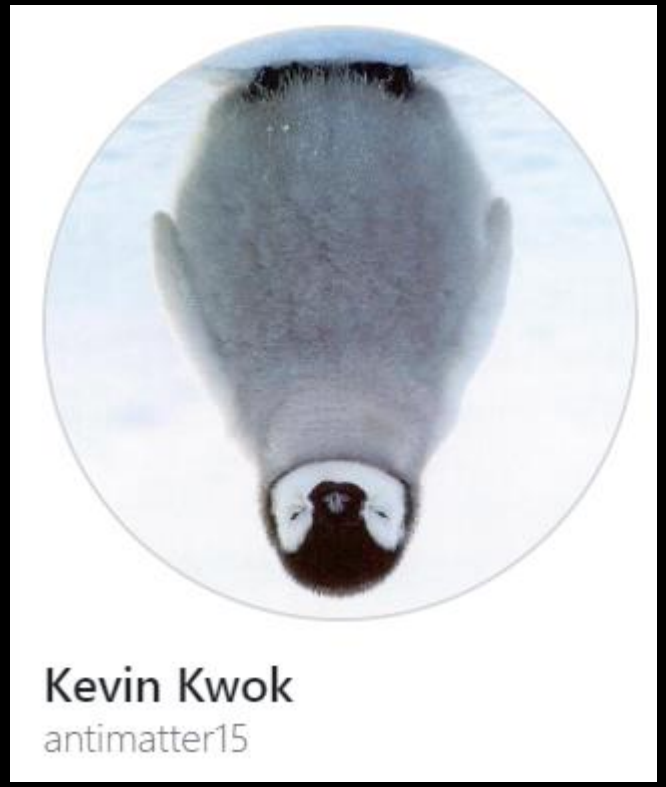
- LEARNING GAUSSIAN SPLATTING! ✓
- LEARNING NEW C++ FEATURES!
- HAVE FUN!

The plan

- Find an implementation of a real-time viewer
- Write it in C++ to learn new features and programming techniques
- Make a presentation about it, to share and inspire!
- Have fun!

Reference implementation

Thank you Kevin Kwok!



splat Public

WebGL 3D Gaussian Splat Viewer

JavaScript 1.7k 169

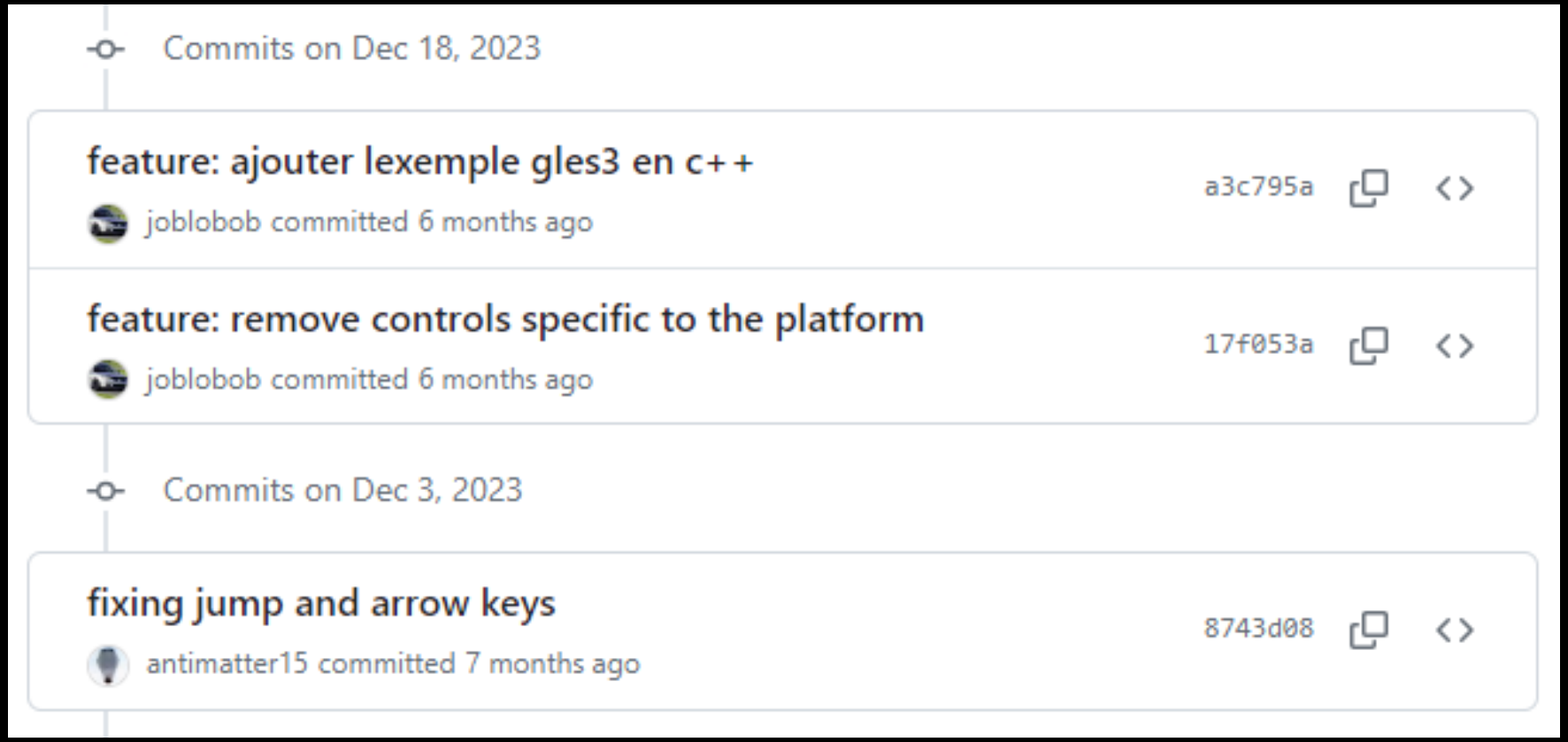
Reference implementation

```
// Camera and view stuff
let camera = {…
};

let viewMatrix = […
];

function getProjectionMatrix(fx, fy, width, height) {…
}
function getViewMatrix(camera) {…
}

// Matrix transformations
function multiply4(a, b) {…
}
function invert4(a) {…
}
function rotate4(a, rad, x, y, z) {…
}
function translate4(a, x, y, z) {…
}

// Data parsing, Texture generatation and sorting splats
function createWorker(self) {…
}

// OpenGL Code
const vertexShaderSource = `…
}`;

const fragmentShaderSource = `…
}`;

// Main execution loop!
async function main() {…
}
```

```
1   // 3D Gaussian Splat viewer Reference Implementation in Javascript/WebGL
2   // https://github.com/antimatter15/splat
3
4   // Camera and view stuff
5   let camera = {…
20  };
21
22  let viewMatrix = […
27  ];
28
29  function getProjectionMatrix(fx, fy, width, height) {…
38  }
39  function getViewMatrix(camera) {…
54  }
55
56  // Matrix transformations
57  function multiply4(a, b) {…
76  }
77  function invert4(a) {…
111 }
```

The story begins

- First commit!



The story begins

- Start from the Qt OpenGL ES Example

The OpenGL Example Window

```
class GLWindow : public QOpenGLWindow {
   Q_OBJECT

public:
   void initializeGL();
   void resizeGL(int w, int h);
   void paintGL();
```

The OpenGL Example Window

```
class GLWindow : public QOpenGLWindow {

private:
   QOpenGLTexture* m_texture        = nullptr;
   QOpenGLShaderProgram* m_program = nullptr;
   QOpenGLBuffer* m_vbo             = nullptr;
   QOpenGLVertexArrayObject* m_vao = nullptr;
```

The OpenGL Example Window

```
class GLWindow : public QOpenGLWindow {

private:
    Logo m_logo;
    QMatrix4x4 m_proj;
    QMatrix4x4 m_world;
    QVector3D m_eye;
    QVector3D m_target = { 0, 0, -1 };
};
```

First Function In C++ !

```
// JS
function getProjectionMatrix(fx, fy, width, height) {
    const znear = 0.2;
    const zfar = 200;
    return [
        [(2 * fx) / width, 0, 0, 0],
        [0, -(2 * fy) / height, 0, 0],
        [0, 0, zfar / (zfar - znear), 1],
        [0, 0, -(zfar * znear) / (zfar - znear), 0],
    ].flat();
}
```

First Function In C++ !

```
// C++
std::vector<float> getProjectionMatrix(float fx, float fy, int
   constexpr float znear = 0.2f;
   constexpr float zfar = 200;
   return {
      (2.f * fx) / width, 0.f, 0.f, 0.f,
       0.f, -(2 * fy) / height, 0.f, 0.f,
       0.f, 0.f, zfar / (zfar - znear), 1.f,
       0.f, 0.f, -(zfar * znear) / (zfar - znear), 0.f
   };
}
```

Choices for data

```
// JS

var texwidth = 1024 * 2;
var texheight = Math.ceil((2 * splatCount) / texwidth);

var texdata   = new Uint32Array(texwidth * texheight * 4);
var texdata_c = new Uint8Array(texdata.buffer);
var texdata_f = new Float32Array(texdata.buffer);
```

Choices for data

```
// C++

int texwidth = 1024 * 2;
int texheight = std::ceil((2 * splatCount) / texwidth);

std::vector<unsigned int>  texdata(texwidth * texheight * 4);
std::vector<unsigned char> texdata_c(texdata.size() * 4);
std::vector<float>         texdata_f(texdata.size());
```

First thing I learned!

```
#include <fstream>

auto length = std::filesystem::file_size("plush.splat");

std::vector<unsigned char> buffer(length);

std::ifstream inputFile("plush.splat", std::ios_base::binary);

inputFile.read(reinterpret_cast<char*>(buffer.data()),length);
inputFile.close();
```

First thing I learned!

```
#include <fstream>

auto length = std::filesystem::file_size("plush.splat");

std::vector<unsigned char> buffer(length);

std::basic_ifstream<unsigned char> inputFile("plush.splat",
                                             std::ios_base::binary);
inputFile.read(buffer.data(), length);
inputFile.close();
```

Then it gets weirder ?!?

```
// OpenGL Stuff
f->glBindBuffer(GL_ARRAY_BUFFER, m_vertexBuffer.bufferId());
f->glBufferData(GL_ARRAY_BUFFER, 32, triangleVertices.data(),
GL_STATIC_DRAW);

const int a_position = m_program.attributeLocation("position");
f->glEnableVertexAttribArray(a_position);
f->glBindBuffer(GL_ARRAY_BUFFER, m_vertexBuffer.bufferId());
f->glVertexAttribPointer(a_position, 2, GL_FLOAT, GL_FALSE, 0, nullptr);

f->glBindTexture(GL_TEXTURE_2D, m_texture.textureId());
auto u_textureLocation = m_program.uniformLocation("u_texture");
f->glUniform1i(u_textureLocation, 0);
```

```
S 3.0.
 == QOpenGLContext::LibGL) {
xt");

eProfile);
```

Ln: 35 Ch: 24 Col: 30 TABS CRLF

0 0

CPU (% of all processors)

100 100

0 0

Summary Events Memory Usage CPU Usage

Events

Show Events (1 of 1)

Memory Usage

Take Snapshot

Enable heap profiling (affects performance)

CPU Usage

Record CPU Profile

Parallel Stacks Diagnostic Tools

Output

Show output from: Debug

```
'hellogles3.exe' (Win32): Loaded 'C:\Windows\System32\userenv.dll'. Symbol loading disabled by Include/Exclude setting.
'hellogles3.exe' (Win32): Loaded 'C:\Windows\System32\authz.dll'. Symbol loading disabled by Include/Exclude setting.
'hellogles3.exe' (Win32): Loaded 'C:\Windows\System32\netapi32.dll'. Symbol loading disabled by Include/Exclude setting.
'hellogles3.exe' (Win32): Loaded 'C:\Windows\System32\version.dll'. Symbol loading disabled by Include/Exclude setting.
'hellogles3.exe' (Win32): Loaded 'C:\Windows\System32\winmm.dll'. Symbol loading disabled by Include/Exclude setting.
'hellogles3.exe' (Win32): Loaded 'C:\Windows\System32\cryptbase.dll'. Symbol loading disabled by Include/Exclude setting.
'hellogles3.exe' (Win32): Loaded 'C:\Windows\System32\srvcli.dll'. Symbol loading disabled by Include/Exclude setting.
'hellogles3.exe' (Win32): Loaded 'C:\Windows\System32\netutils.dll'. Symbol loading disabled by Include/Exclude setting.
'hellogles3.exe' (Win32): Loaded 'C:\Windows\System32\imm32.dll'. Symbol loading disabled by Include/Exclude setting.
'hellogles3.exe' (Win32): Loaded 'C:\Windows\System32\bcryptprimitives.dll'. Symbol loading disabled by Include/Exclude setting.
'hellogles3.exe' (Win32): Loaded 'C:\Windows\System32\SHCore.dll'. Symbol loading disabled by Include/Exclude setting.
'hellogles3.exe' (QML): Connecting to the QML runtime...
```

	Type
	QGuiApplication
	int
	char * *
	QSurfaceFormat
> m_progr...	GLWindow
> m_progr...	GLWindowSplat

Developer PowerShell Code Metrics Results Call Stack Breakpoints Exception Settings Command Window Immediate Window Output Error List

0 12 Detached at 66e80178 splatty

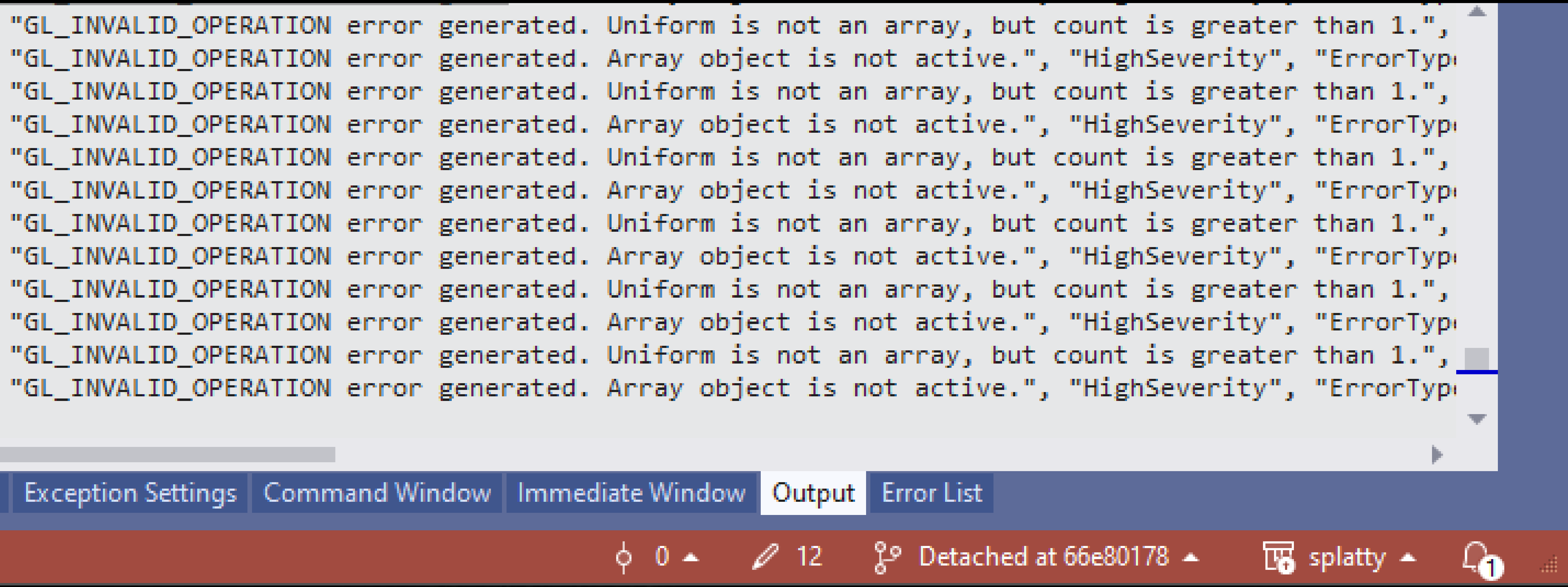
Debugging OpenGL calls

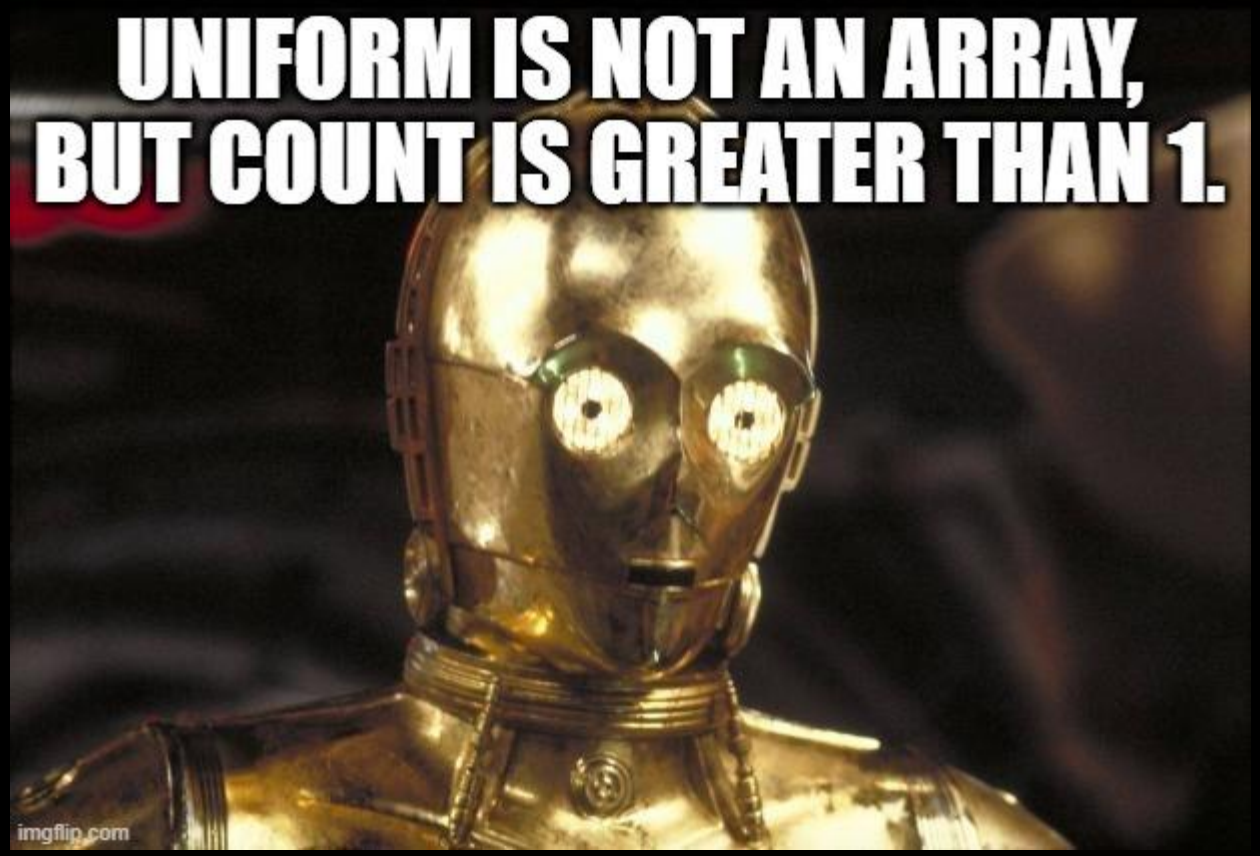
```
\DataExchange.dll'. Symbol loading disabled by Include/Exclude setting.
\dcomp.dll'. Symbol loading disabled by Include/Exclude setting.



, "Buffer detailed info: Buffer object 1 (bound to GL_ARRAY_BUFFER_ARB, usage hint is GL_STATI
"GL_INVALID_OPERATION error generated. Invalid VAO/VBO/pointer usage.", "HighSeverity", "Error
"GL_INVALID_OPERATION error generated. Invalid VAO/VBO/pointer usage.", "HighSeverity", "Error
"GL_INVALID_OPERATION error generated. Invalid VAO bound.", "HighSeverity", "ErrorType")
"GL_INVALID_OPERATION error generated. Uniform is not an array, but <count> is greater than 1.
"GL_INVALID_OPERATION error generated. Uniform is not an array, but <count> is greater than 1.
"GL_INVALID_OPERATION error generated. Uniform is not an array, but count is greater than 1.",
```

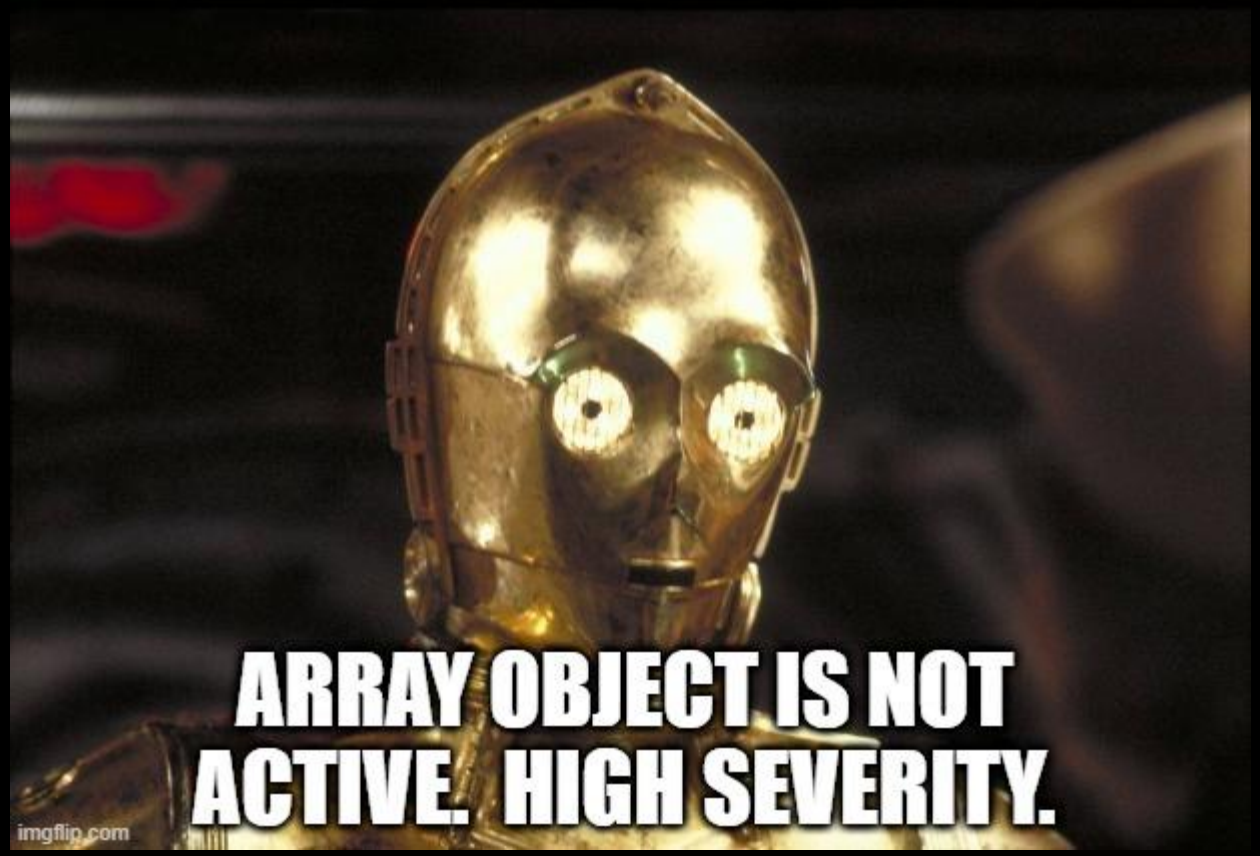
Exception Settings | Command Window | Immediate Window | Output | Error List

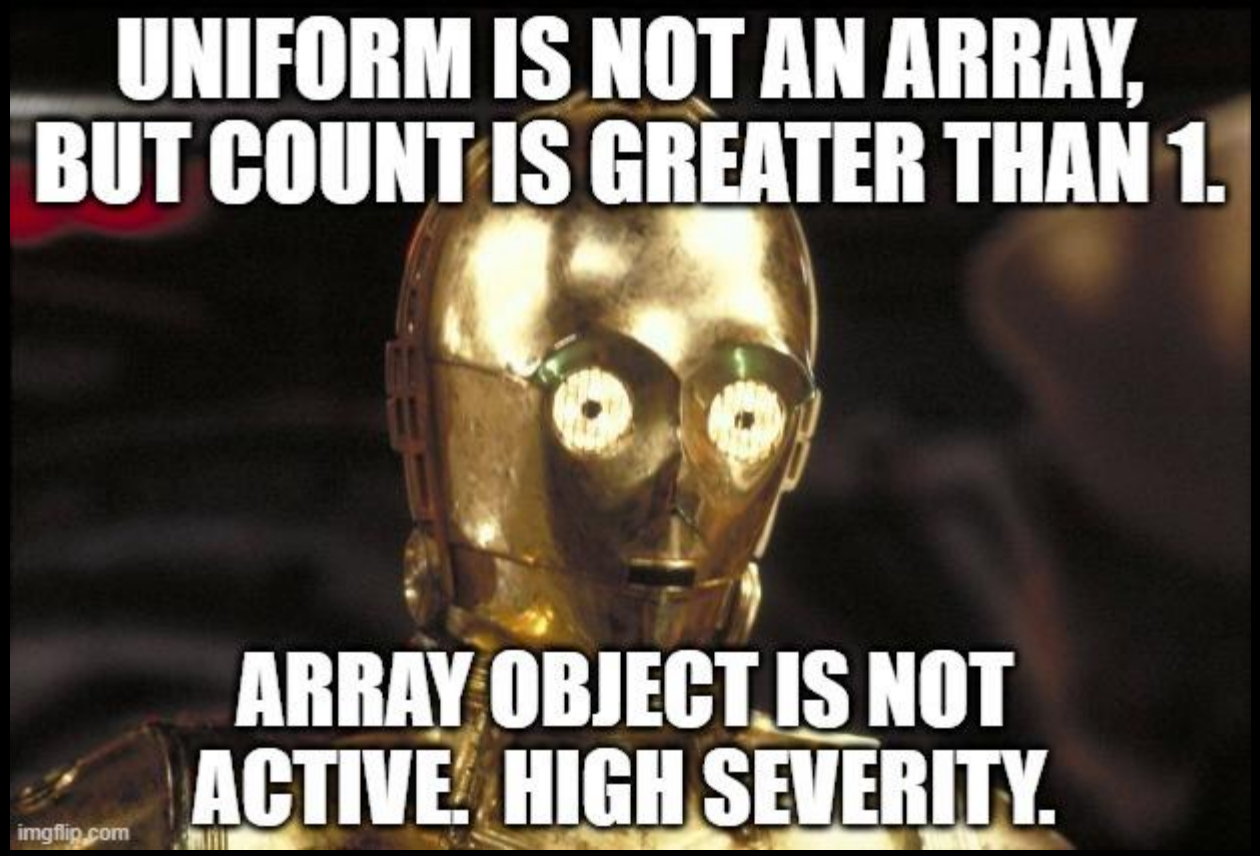
0 | 12 | Detached at 66e80178 | splatty | 1

Debugging OpenGL calls




UNIFORM IS NOT AN ARRAY,
BUT COUNT IS GREATER THAN 1.
imgflip.com

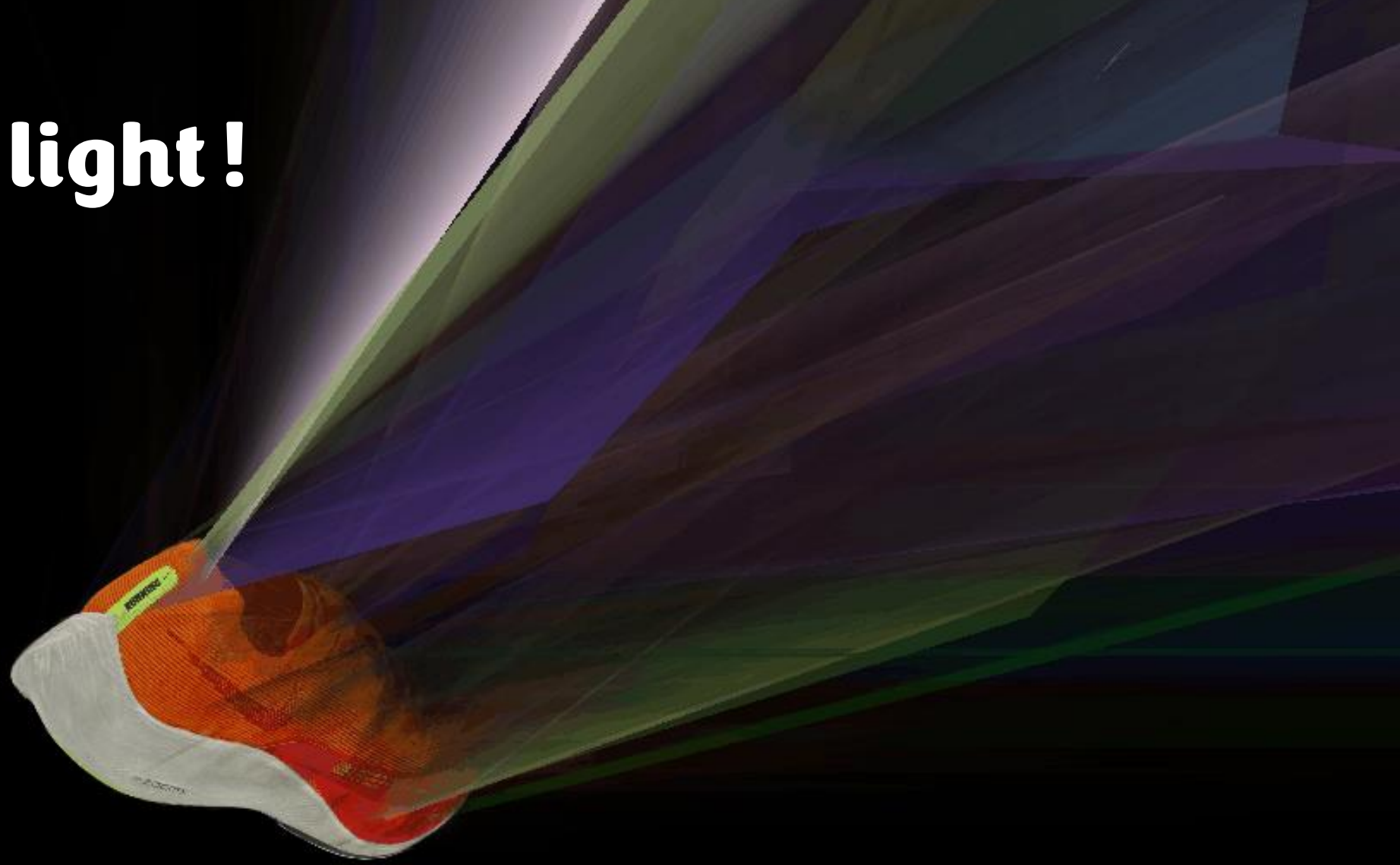

ARRAY OBJECT IS NOT
ACTIVE. HIGH SEVERITY.
imgflip.com


UNIFORM IS NOT AN ARRAY,
BUT COUNT IS GREATER THAN 1.
ARRAY OBJECT IS NOT
ACTIVE. HIGH SEVERITY.
imgflip.com

First light !

First light !

Real first light !



Second time!

~~Second~~ time!

~~Second~~ Third time!

~~Second~~ ~~Third~~ time!

~~Second~~ ~~Third~~ fourth time!

The code has been splatted!

```
1   // 3D gaussian splatting viewer rewrite in C++
2   // let's get crazy and copy everything here!
3
4   struct position {···
6   };
7   struct rotation {···
9   };
10  struct camera {···
15  };
16
17  static camera baseCamera = { .id = 0, .width = 1024, .height = 728, ···
21  static const std::vector<float> viewMatrix = { 0.47, 0.04, 0.88, 0, -0.11, 0.99, 0.02, 0, -0.88, -0.11, 0.47, 0, 0.07, 0.03, 6.55, 1 };
22
23  static std::vector<float> getProjectionMatrix(float fx, float fy, int width, int height)
24  {···
44  }
45
46  static std::vector<float> multiply4(const std::vector<float>& a, const std::vector<float>& b)
47  {···
66  }
67  static std::vector<float> invert4(const std::vector<float>& a)
68  {···
101 }
102 static std::vector<float> rotate4(const std::vector<float>& a, float rad, float x, float y, float z)
103 {···
136 }
137 static std::vector<float> translate4(const std::vector<float>& a, float x, float y, float z)
138 {···
155 }
156
157
158 class GLWindowSplat : public QOpenGLWindow
159 {···
162 public:···
377 private slots:···
380 private:···
405 };
```

```
1   // 3D gaussian splatting viewer rewrite in C++
2   // let's get crazy and copy everything here!
3
4   struct position { ···
6   };
7   struct rotation { ···
9   };
10  struct camera { ···
15  };
16
17  static camera baseCamera = { .id = 0, .width = 1024, .height = 728, ···
21  static const std::vector<float> viewMatrix = { 0.47, 0.04, 0.88, 0, -0.11, 0.99, 0
22
23  static std::vector<float> getProjectionMatrix(float fx, float fy, int width, int h
24  { ···
44  }
45
46  static std::vector<float> multiply4(const std::vector<float>& a, const std::vector
47  { ···
66  }
```

What’s next?

Modules!

```
// CMakeLists.txt

add_library(splatlib)
target_sources(splatlib PUBLIC
   FILE_SET my_modules TYPE CXX_MODULES FILES
      "splatty.ixx")

target_link_libraries(splatty PRIVATE
   splatlib)
```

Modules!

```
/*
* Yes, this is a C++ module! I love it!
*/

module;

export module splatty;
```

Modules!

```
// GLWindow-Splat.cpp

#include <print>
#include <vector>

import splatty;

// ...
```

Modules, one big one!

```
1  /*
2  * Yes this is a C++ module! I love it!
3  * It's intended to be a single file for the whole implementation for the moment,
4  */
5
6
7  module;
8
9  #include <cmath>
10 #include <limits>
11 #include <print>
12 #include <vector>
13
14 export module splatty;
15
16 export static std::vector<float> viewMatrix = { 0.47, 0.04, 0.88, 0, -0.11, 0.99,
17
18 export constexpr int focalWidth  = 1500;
19 export constexpr int focalHeight = 1500;
20
```

Modules, one by one!

```
// CMakeLists.txt

add_library(splatlib)
target_sources(splatlib PUBLIC
   FILE_SET my_modules TYPE CXX_MODULES FILES
      splatmath.ixx
      splatopengl.ixx
      splatdata.ixx
      splatreader.ixx
      splatshaders.ixx
      splatcoroutine.ixx
      splatty.ixx)
```

Multi Modular Mastery!

```
/*
* Yes this is a C++ module! I love it!
*
* It's not a single file anymore, multi modular mastery mhuhuhahaha!
*/

module;

#include <filesystem>
#include <ranges>
#include <vector>

export module splatty;

import splat.coroutine;
import splat.data;
import splat.opengl;
import splat.math;
import splat.reader;

```

What's left in Splatty?

```
Splatty(const std::filesystem::path& path) {

    std::vector<unsigned char> data = Splat::readFromFile(path);

    splatCount = data.size() / rowLength;
    depthIndex.resize(splatCount);

    m_data     = std::make_unique<SplatData>(data);
    m_gl       = std::make_unique<glsplat>(splatCount);
}
```

What's left in Splatty?

```
void setView(float x, float y, float z) {
   float dot = prevX * x + prevY * y + prevZ * z;
   if (std::abs(dot - 1) > 0.01f) {
      std::vector<unsigned int> texdata = generateTexture();
      m_gl->setTextureData(texdata, texwidth, texheight);

      std::vector<unsigned int> depthIndex = sortByDepth(x,y,z);
      m_gl->setDepthIndex(depthIndex);
   }

   m_gl->viewChanged();
}
```

And in other Modules?

OpenGL Module!

```
1  /*
2  * Yes this is a C++ module! I love it!
3  */
4
5  module;
6
7  #include <QOpenGLBuffer>
8  #include <QOpenGLContext>
9  #include <QOpenGLExtraFunctions>
10 #include <QOpenGLShaderProgram>
11 #include <QOpenGLTexture>
12 #include <QOpenGLVertexArrayObject>
13
14 export module splat.opengl;
15
16 import splat.math;
17 import splat.shaders;
18
19 export static std::vector<float> viewMatrix = { 0.47, 0.04, 0.88, 0, -0.11, 0.99,
20
```

Shader Module!

```
module;

#include <string>

export module splat.shaders;

export namespace ShaderSource {
   constexpr auto vertex   = "#version 330…\n";
   constexpr auto fragment = "#version 330…\n";
};
```

Vertex Shader

```
  constexpr auto vertex = "#version 330\n"
   "#extension GL_ARB_shading_language_packing : enable\n"
   "uniform highp usampler2D u_texture;\n"
   "uniform mat4 projection, view;\n"
   "uniform vec2 focal, viewport;\n“
   "in vec2 position;\n"
   "in int index;\n"
   "out vec4 vColor, vPosition;\n"
   "void main () {\n"
   " uvec4 cen = texelFetch(u_texture, ivec2((uint(index) &
0x3ffu) << 1, uint(index) >> 10), 0);\n"
   "  vec4 cam = view * vec4(uintBitsToFloat(cen.xyz), 1);\n"
   "  vec4 pos2d = projection * cam;\n"
```

Vertex Shader

```
  constexpr auto vertex = "#version 330\n"
   "#extension GL_ARB_shading_language_packing : enable\n"
   "uniform highp usampler2D u_texture;\n"
   "uniform mat4 projection, view;\n"
   "uniform vec2 focal, viewport;\n“
   "in vec2 position;\n"
   "in int index;\n"
   "out vec4 vColor, vPosition;\n"
   "void main () {\n"
   " uvec4 cen = texelFetch(u_texture, ivec2((uint(index) &
0x3ffu) << 1, uint(index) >> 10), 0);\n"
   "  vec4 cam = view * vec4(uintBitsToFloat(cen.xyz), 1);\n"
   "  vec4 pos2d = projection * cam;\n"
```

Vertex Shader

```
  constexpr auto vertex = "#version 330\n"
   "#extension GL_ARB_shading_language_packing : enable\n"
   "uniform highp usampler2D u_texture;\n"
   "uniform mat4 projection, view;\n"
   "uniform vec2 focal, viewport;\n“
   "in vec2 position;\n"
   "in int index;\n"
   "out vec4 vColor, vPosition;\n"
   "void main () {\n"
   " uvec4 cen = texelFetch(u_texture, ivec2((uint(index) &
0x3ffu) << 1, uint(index) >> 10), 0);\n"
   "  vec4 cam = view * vec4(uintBitsToFloat(cen.xyz), 1);\n"
   "  vec4 pos2d = projection * cam;\n"
```

Fragment Shader

```
constexpr auto fragment = "#version 330\n"
        "in highp vec4 vColor;\n"
        "in highp vec2 vPosition;\n"
        "out highp vec4 fragColor;\n"
        "void main () {\n"
        "  highp float A = -dot(vPosition, vPosition);\n"
        "  if (A < -4.0) discard;\n"
        "  highp float B = exp(A) * vColor.a;\n"
        "  fragColor = vec4(B * vColor.rgb, B);\n"
        "}\n";
```

Fragment Shader

```
constexpr auto fragment = "#version 330\n"
        "in highp vec4 vColor;\n"
        "in highp vec2 vPosition;\n"
        "out highp vec4 fragColor;\n"
        "void main () {\n"
        "  highp float A = -dot(vPosition, vPosition);\n"
        "  if (A < -4.0) discard;\n"
        "  highp float B = exp(A) * vColor.a;\n"
        "  fragColor = vec4(B * vColor.rgb, B);\n"
        "}\n";
```

Shader Module!

```
module;

#include <string>

export module splat.shaders;

export namespace ShaderSource {
   constexpr auto vertex   = "#version 330…\n";
   constexpr auto fragment = "#version 330…\n";
};
```

Embedding Shaders!

```
module;

#include <embed>

export module splat.shaders;
// Please Put My Data In My Executable
// And Stop Doing Crazy Nonsense, The Feature!
export namespace ShaderSource {
   constexpr auto vertex  = std::embed("vertexShader.vert");
   constexpr auto fragment= std::embed("fragmentShader.frag");
};
```

Nah! Embedding Shaders!

```
module;

#include <battery/embed.hpp>

export module splat.shaders;
// Please Put My Data In My Executable
// And Stop Doing Crazy Nonsense, The Feature!
export namespace ShaderSource {
   constexpr auto vertex  = b::embed<"vertexShader.vert">();
   constexpr auto fragment= b::embed<"fragmentShader.frag">();
};
```

Embed everything!

```
module;

#include <battery/embed.hpp>

export module splat.data;
// Please Put My Data In My Executable
// And Stop Doing Crazy Nonsense, The Feature!
export constexpr auto plushData = b::embed<"plush.splat">();
```

Embed everything!

```
module;

#include <battery/embed.hpp>

export module splat.data;
// Please Put My Data In My Executable
// And Stop Doing Crazy Nonsense, The Feature!
export constexpr auto plushData = b::embed<"plush.splat">();
```

Embed everything!

Out of Memory!

Next stop?

No more loops!! Only algos!! (yeah right...)

First try, hmm.....

```
for (int i = 0; i < M.size(); i++) {
   M[i] = M[i] * scale[std::floor(i / 3)];
}
```

Ranges Transform

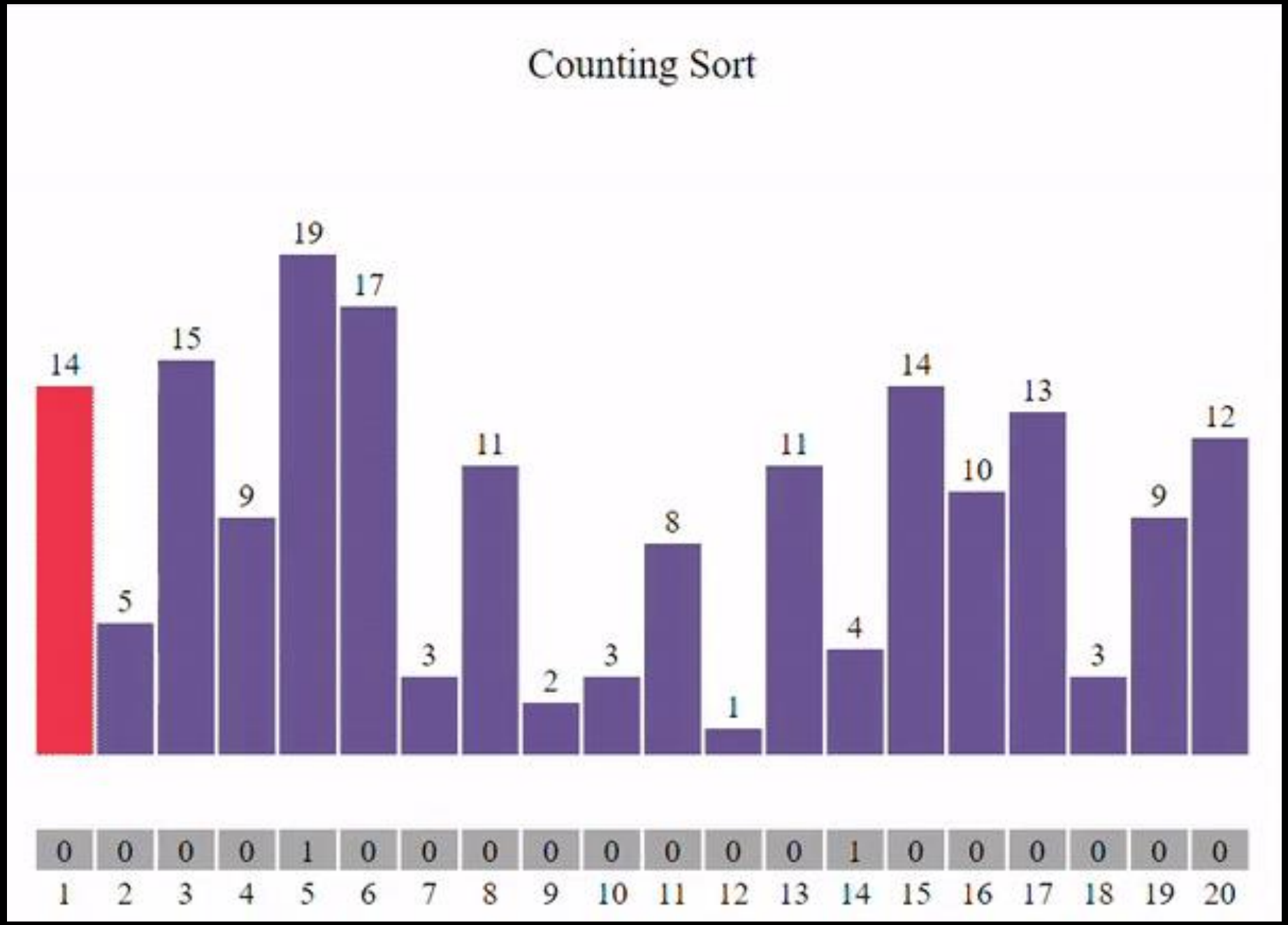
```
for (int i = 0; i < M.size(); i++) {
   M[i] = M[i] * scale[std::floor(i / 3)];
}
```

```
std::ranges::transform(M, scale, M.begin(), multiplies<float>());
```

More Ranges Transform

```
std::vector<unsigned int> counts0(256 * 256);

for (int i = 0; i < splatCount; i++) {
   depth[i] = std::floor((depth[i]-minDepth)*depthInv);
   counts0[depth[i]]++; //count occurences
}
```

Counting Sort



More Ranges Transform

```
std::vector<unsigned int> counts0(256 * 256);

for (int i = 0; i < splatCount; i++) {
   depth[i] = std::floor((depth[i]-minDepth)*depthInv);
   counts0[depth[i]]++; //count occurences
}
```

More Ranges Transform

```
auto count(unsigned int val) {
   unsigned int index = std::floor((val-minDepth) * depthInv);
   counts0[index]++; //count occurences
   return index;
};

for (int i = 0; i < splatCount; i++) {
   depth[i] = count(depth[i]);
}
```

From Ranges Transform

```
auto count = [&](unsigned int val) {
   unsigned int index = std::floor((val-minDepth) * depthInv);
   counts0[index]++; //count occurences
   return index;
};

std::ranges::transform(depth, depth.begin(), count);
```

To Views Transform!

```
auto count = [&](unsigned int val) {
   unsigned int index = std::floor((val-minDepth) * depthInv);
   counts0[index]++; //count occurences
   return index;
};

depth  | std::views::transform(count)
       | std::ranges::to<std::vector>();
```

Fun with views!

```
for (int i = 0; i < splatCount; i++) {

    std::memcpy(&texdata[8 * i + 0], &buffer[8 * i + 0], 12);
    // ...
}
```

Fun with views and striiiiiiiiide!

```
for (auto i : std::views::iota(0u, buffer.size()) |
              std::views::stride(8)) {
   std::memcpy(&texdata[i], &buffer[i], 12);
   // ...
}
```

Progressive display of splats!

Conversions and casting?

```
constexpr auto to_uint = [](float v) {
   return std::bit_cast<unsigned int>(v);
};

const auto uintBuffer =
    fbuffer | std::views::transform(to_uint)
            | std::ranges::to<std::vector<unsigned int>>();
```

Data structure work

```
void GLWindowSplat::worldInteraction(std::vector<float>& view)
{
   auto inv = invert4(view);

   inv = rotate4(inv, std::sin(16.0f / 2000.5f), 1, -1, 1);
   inv = translate4(inv, 0.05, 0.05, -0.5);

   view = invert4(inv);
}
```

Data structure work

```
void GLWindowSplat::worldInteraction(std::array<float, 16>& view)
{
   auto inv = invert4(view);

   inv = rotate4(inv, std::sin(16.0f / 2000.5f), 1, -1, 1);
   inv = translate4(inv, 0.05, 0.05, -0.5);

   view = invert4(inv);
}
```

MD SPAN!

```
using Matrix4 = std::mdspan<float, std::extents<int, 4, 4> >;
void GLWindowSplat::worldInteraction(Matrix4 view)
{
   auto inv = invert4(view);

   inv = rotate4(inv, std::sin(16.0f / 2000.5f), 1, -1, 1);
   inv = translate4(inv, 0.05, 0.05, -0.5);

   view = invert4(inv);
}
```

MD SPAN!

```
void translate4(std::array<float, 16>& a, float x, float y, float

   a[12] += a[0] * x + a[4] * y + a[8] * z;
   a[13] += a[1] * x + a[5] * y + a[9] * z;
   a[14] += a[2] * x + a[6] * y + a[10] * z;
   a[15] += a[3] * x + a[7] * y + a[11] * z;

}
```

MD SPAN!

```
void translate4(Matrix4 a, float x, float y, float z) {

   a[3, 0] += a[0, 0] * x + a[1, 0] * y + a[2, 0] * z;
   a[3, 1] += a[0, 1] * x + a[1, 1] * y + a[2, 1] * z;
   a[3, 2] += a[0, 2] * x + a[1, 2] * y + a[2, 2] * z;
   a[3, 3] += a[0, 3] * x + a[1, 3] * y + a[2, 3] * z;

}
```

MD SPAN, the reality

```
void translate4(Matrix4 a, float x, float y, float z) {

   a[std::array{3, 0}] += a[std::array{0, 0}] * x + ...
   a[std::array{3, 1}] += a[std::array{0, 1}] * x + ...
   a[std::array{3, 2}] += a[std::array{0, 2}] * x + ...
   a[std::array{3, 3}] += a[std::array{0, 3}] * x + ...

}
```

Std array and mdspan everywhere!

```
std::array<float, 16> defaultViewMatrix =
      {   0.47, 0.04, 0.88, 0,
         -0.11, 0.99, 0.02, 0,
         -0.88,-0.11, 0.47, 0,
          0.07, 0.03, 6.55, 1
      };
```

Std array and mdspan everywhere!

```
std::array<float, 16> defaultViewMatrix =
      {   0.47, 0.04, 0.88, 0,
         -0.11, 0.99, 0.02, 0,
         -0.88,-0.11, 0.47, 0,
          0.07, 0.03, 6.55, 1
      };

auto viewMatrix = std::mdspan(defaultViewMatrix.data(), 4, 4);
```

Passing mdspan to the opengl layer!

```
auto viewMatrix = std::mdspan(defaultViewMatrix.data(), 4, 4);

f->glUniformMatrix4fv(viewLoc, 1, false,
                      viewMatrix.data_handle());
```

Passing mdspan to the opengl layer!

```
auto viewMatrix = std::mdspan(defaultViewMatrix.data(), 4, 4);

f->glUniformMatrix4fv(viewLoc, 1, false,
                      viewMatrix.data_handle());
```

Passing mdspan around!

```
// std::array<float, 16>
m_splatty.setPosition(
   viewMatrix[2],  // x
   viewMatrix[6],  // y
   viewMatrix[10]  // z
);
```

Passing mdspan around!

```
// std::mdspan<float, 4, 4>
m_splatty.setPosition(
   viewMatrix[0, 2],  // x
   viewMatrix[1, 2],  // y
   viewMatrix[2, 2]   // z
);
```

Passing mdspan around!

```
// std::mdspan<float, 4, 4>
m_splatty.setPosition(
   viewMatrix[std::array{0, 2}], // x
   viewMatrix[std::array{1, 2}], // y
   viewMatrix[std::array{2, 2}]  // z
);
```

Passing mdspan around!

```
// std::mdspan<float, 4, 4>
m_splatty.setPosition(
   viewMatrix[x],   // x = std::array{0, 2}
   viewMatrix[y],   // y = std::array{1, 2}
   viewMatrix[z]    // z = std::array{2, 2}
);
```


Let’s use a bigger file!

WebGL 3D Gaussian Splat Viewer

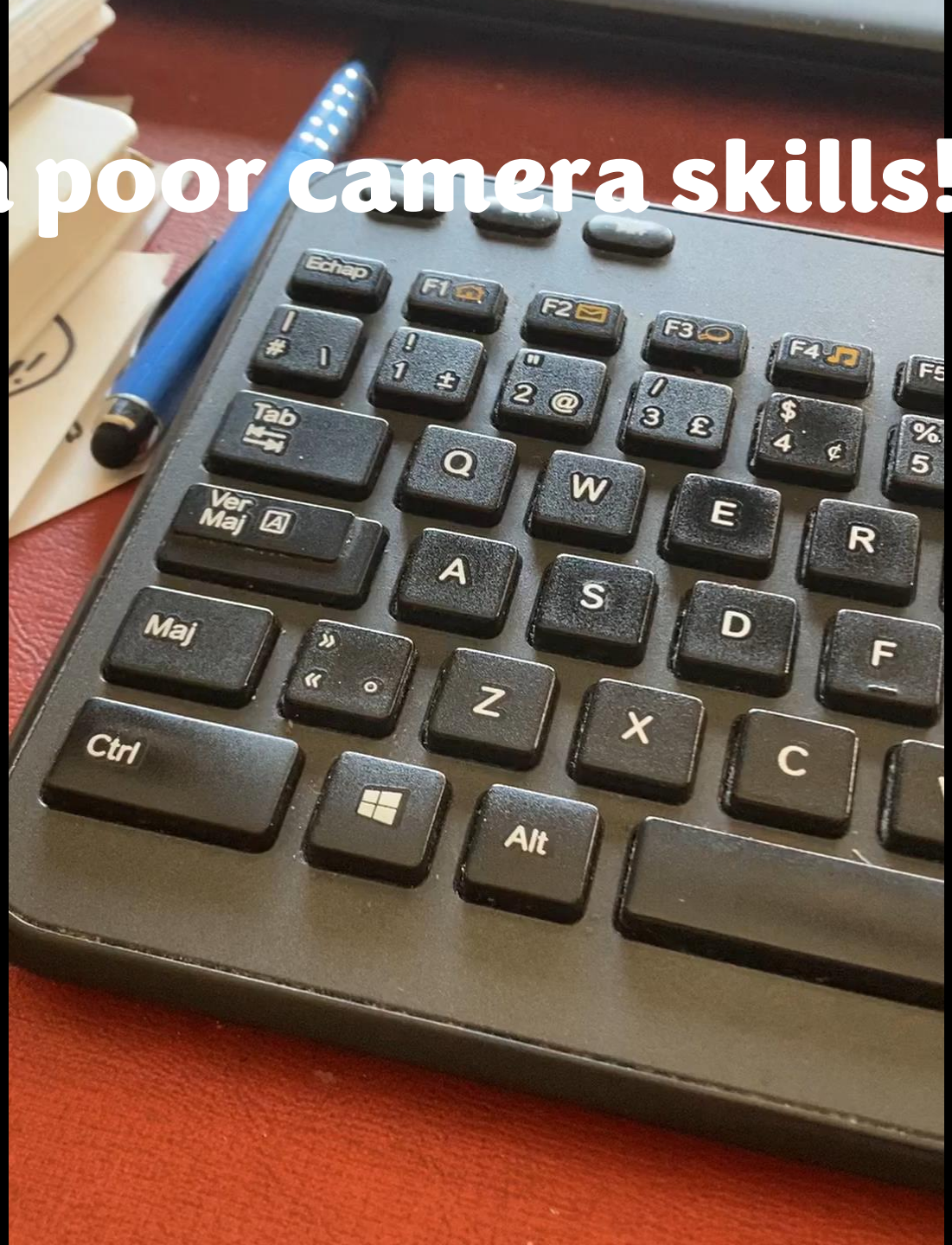
By Kevin Kwok. Code on Github.

▶ Use mouse or arrow keys to navigate.

Back to Splatty!!!

Making your own splats!

Even with poor camera skills!

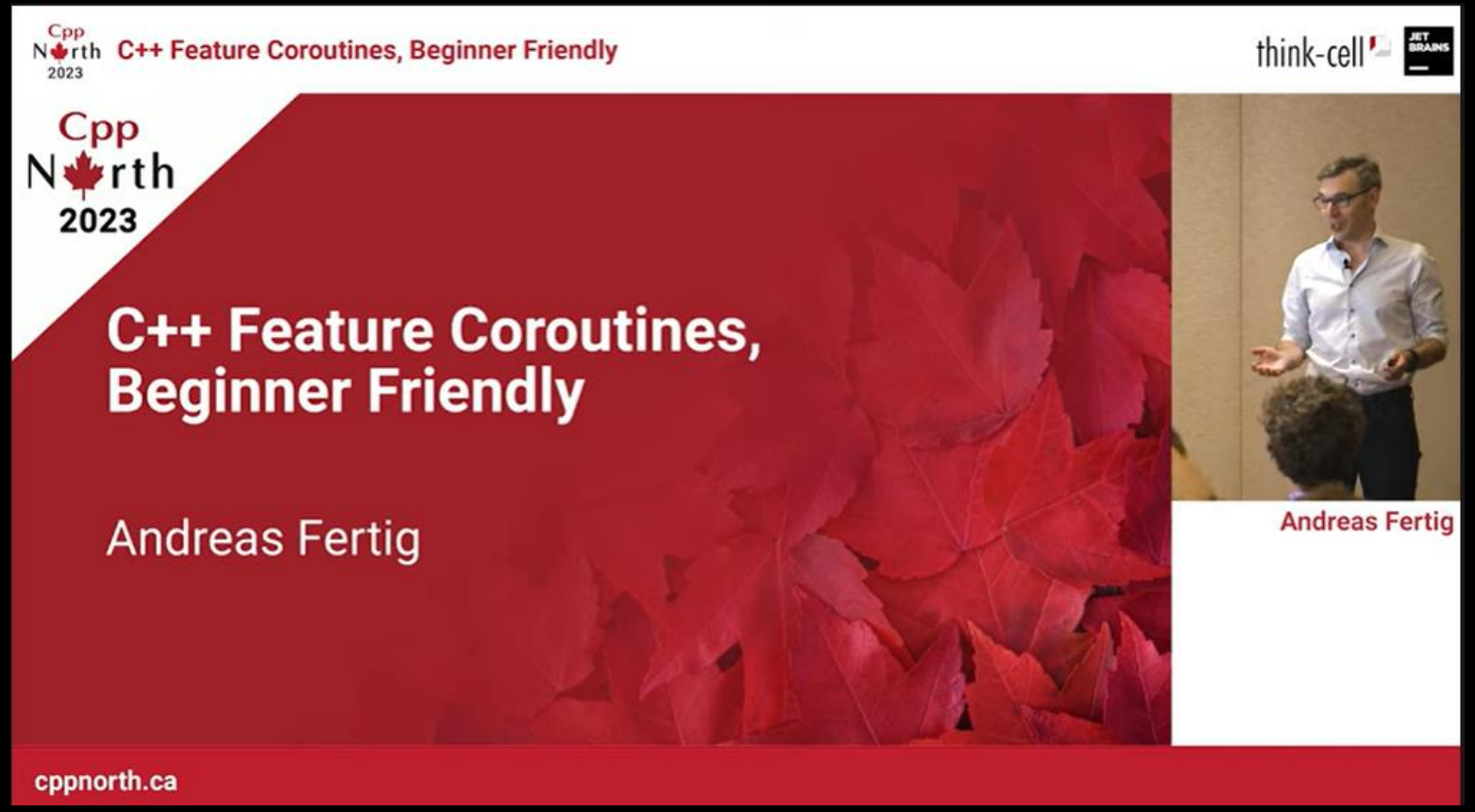


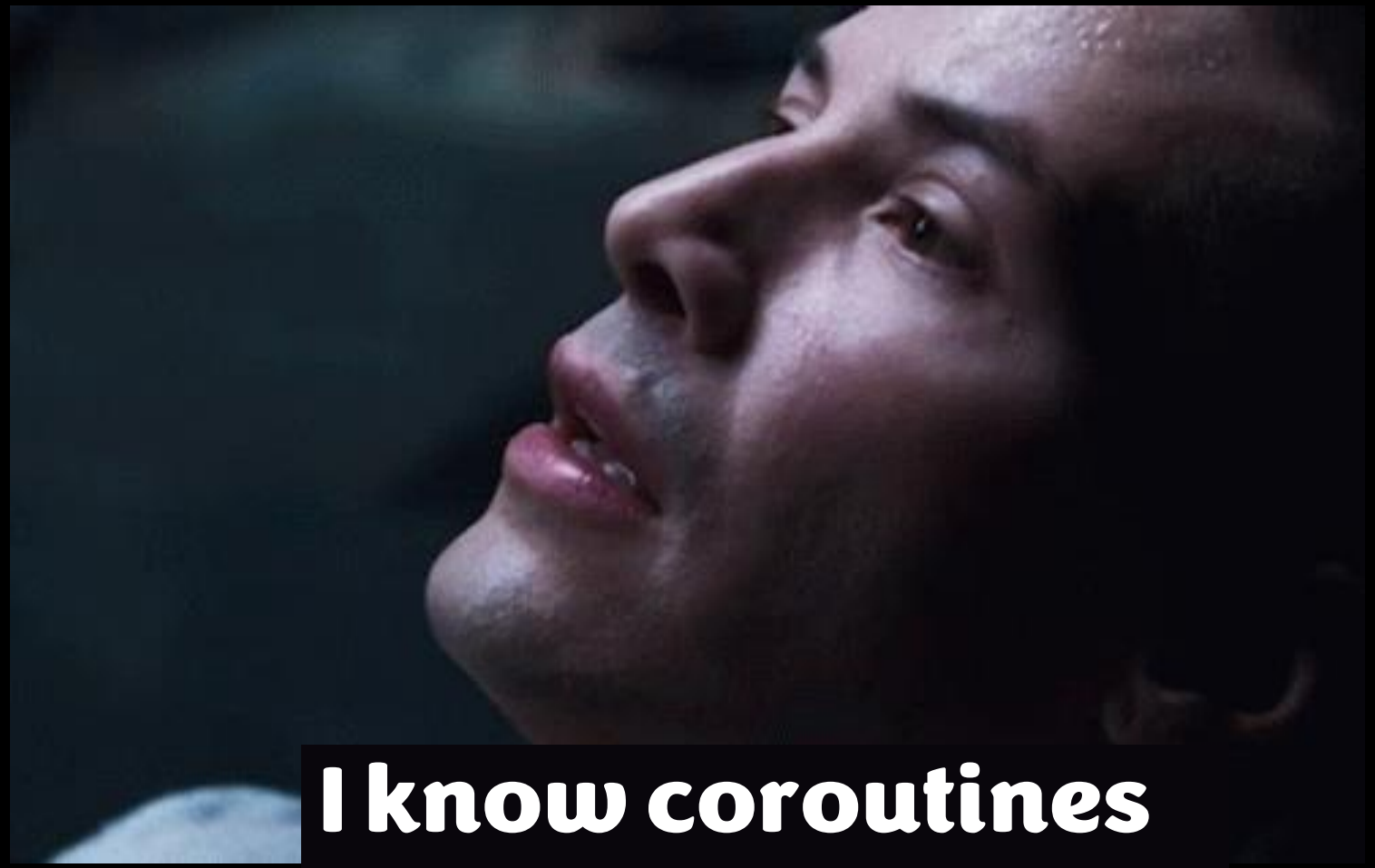
But you can have good results!

Back to Splatty!!!

Coroutines!!!

- **Thank you Andreas Fertig!**




I know coroutines

Coroutines?

Coroutines!!!

```
/* Yes, this is a C++ coroutine module! I love it! */
module;

#include <coroutine>
#include <print>
#include <string>

export module splat.coroutine;

export struct Chat {  // coroutine type!  };

export Chat Fun()  {  // coroutine definition! }
```

The fun coroutine!

```
export Chat Fun() {

   int funLevel = 0;

   co_yield "Hello, the fun can start!";

   while (funLevel < 50) {
      std::println("{}{}", funLevel++, co_await std::string {});

      co_yield "Polo!! %1 " + std::to_string(funLevel);
   }
   co_return "Not fun anymore... (╥﹏╥)";
}
```

The game is on!

```
// splatty.ixx


export struct Splatty {


   void setView(const std::array<float, 16>& newviewProj) {
      viewProj = newviewProj;

      if (std::abs(dot - 1) > 0.01) {

            generateTexture();

            sortByDepth(viewProj[x], viewProj[y], viewProj[z]);
```

The game is on!

```
// splatty.ixx
import splat.coroutine;

export struct Splatty {


   void setView(const std::array<float, 16>& newviewProj) {
      viewProj = newviewProj;

      if (std::abs(dot - 1) > 0.01) {

            generateTexture();

            sortByDepth(viewProj[x], viewProj[y], viewProj[z]);
```

The game is on!

```
// splatty.ixx
import splat.coroutine;

export struct Splatty {
   Chat hiddenGame = Fun(); // Creation of the coroutine

   void setView(const std::array<float, 16>& newviewProj) {
      viewProj = newviewProj;

      if (std::abs(dot - 1) > 0.01) {

            generateTexture();

            sortByDepth(viewProj[x], viewProj[y], viewProj[z]);
```

The game is on!

```
// splatty.ixx
import splat.coroutine;

export struct Splatty {
   Chat hiddenGame = Fun(); // Creation of the coroutine

   void setView(const std::array<float, 16>& newviewProj) {
      viewProj = newviewProj;
      hiddenGame.shout("Marco!!?!");
      if (std::abs(dot - 1) > 0.01) {

            generateTexture();

            sortByDepth(viewProj[x], viewProj[y], viewProj[z]);
```

The game is on!

```
// splatty.ixx
import splat.coroutine;

export struct Splatty {
   Chat hiddenGame = Fun(); // Creation of the coroutine

   void setView(const std::array<float, 16>& newviewProj) {
      viewProj = newviewProj;
      hiddenGame.shout("Marco!!?!");
      if (std::abs(dot - 1) > 0.01) {
            std::println({}, hiddenGame.listen());
            generateTexture();

            sortByDepth(viewProj[x], viewProj[y], viewProj[z]);
```

```
Hello, the fun can start!
```

Hidden marco polo!

The Chat coroutine type

```
export struct Chat {
   struct promise_type {
      std::string msgOut {}, msgIn {}; // Storing data

      Chat get_return_object(){ return Chat { this }; }
      std::suspend_never initial_suspend() { return {}; }
      std::suspend_always final_suspend() { return {}; }
```

The Chat coroutine type

```
export struct Chat {
   struct promise_type {
      std::suspend_always yield_value(std::string msg) {
         msgOut = std::move(msg); // Store value from co_yield
         return {};
      }
      void return_value(std::string msg) {
         msgOut = std::move(msg); // Store value from co_return
      }
      auto await_transform(std::string) {
         std::string awaiter::await_resume() const {
            return std::move(pt.msgIn); // Value from co_await
         }
      return awaiter { *this }; } };
```

The Chat coroutine type

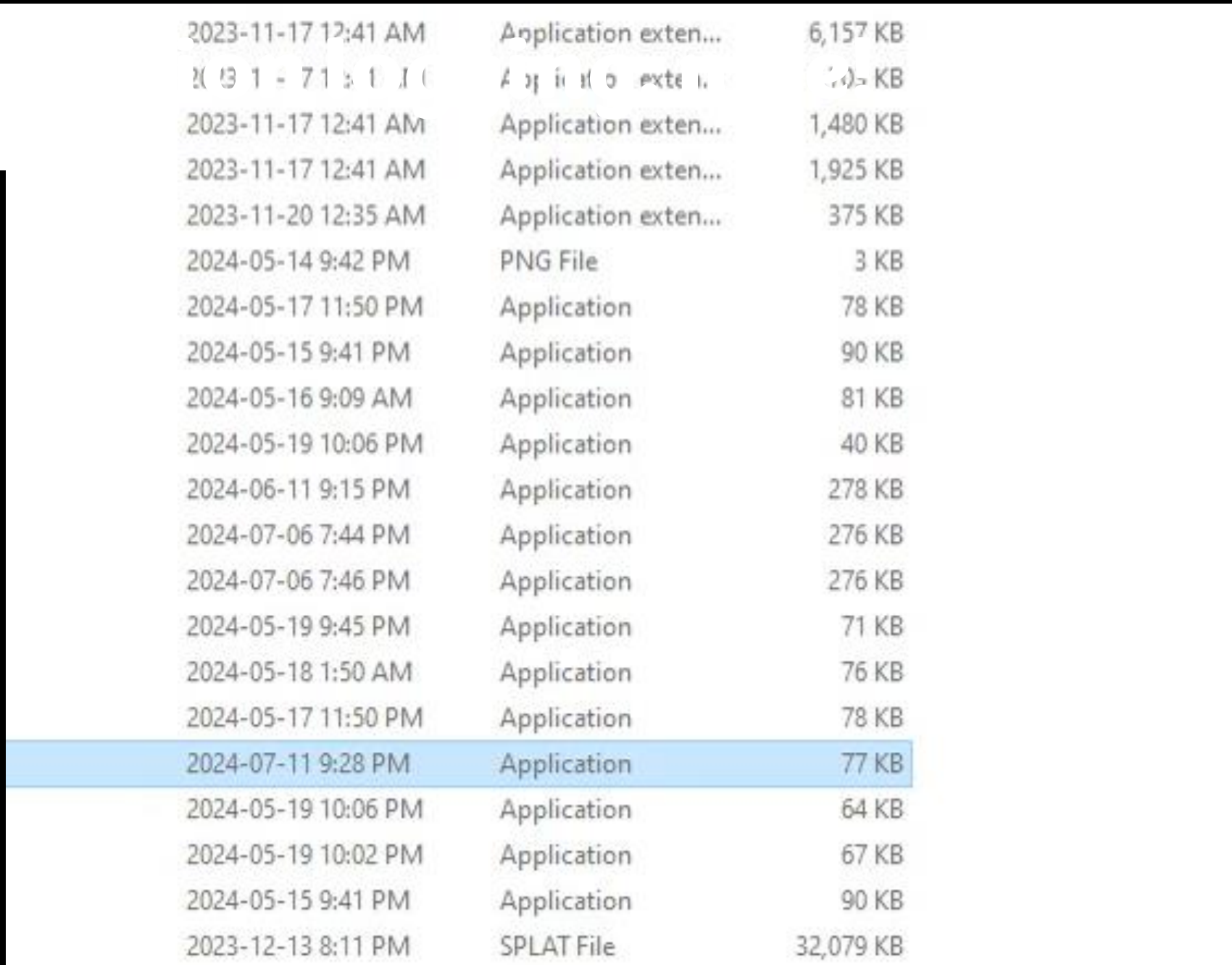
```
export struct Chat {
   std::coroutine_handle<promise_type> co_handle {};

   void shout(std::string message) {
      co_handle.promise().msgIn = std::move(message);
      if (not co_handle.done()) co_handle.resume();
   }
   std::string listen() {
      if (not co_handle.done()) co_handle.resume();
      return std::move(co_handle.promise().msgOut);
   }
};
```

More useful coroutines!

```
CountLogger LoggingCoroutine() {
   int c = 0;
   steady_clock::time_point startPoint = steady_clock::now();
   co_yield "Initialization done!";

   while (c++ < 100) {
      co_yield
         std::to_string(
            std::duration_cast<milliseconds>(
               std::steady_clock::now() - startPoint).count());
   }
   co_return "Finished !";
}
```

Corouti

-



2023-11-17 12:41 AM	Application exten...	6,157 KB
[illegible]	[illegible]	[illegible] KB
2023-11-17 12:41 AM	Application exten...	1,480 KB
2023-11-17 12:41 AM	Application exten...	1,925 KB
2023-11-20 12:35 AM	Application exten...	375 KB
2024-05-14 9:42 PM	PNG File	3 KB
2024-05-17 11:50 PM	Application	78 KB
2024-05-15 9:41 PM	Application	90 KB
2024-05-16 9:09 AM	Application	81 KB
2024-05-19 10:06 PM	Application	40 KB
2024-06-11 9:15 PM	Application	278 KB
2024-07-06 7:44 PM	Application	276 KB
2024-07-06 7:46 PM	Application	276 KB
2024-05-19 9:45 PM	Application	71 KB
2024-05-18 1:50 AM	Application	76 KB
2024-05-17 11:50 PM	Application	78 KB
2024-07-11 9:28 PM	Application	77 KB
2024-05-19 10:06 PM	Application	64 KB
2024-05-19 10:02 PM	Application	67 KB
2024-05-15 9:41 PM	Application	90 KB
2023-12-13 8:11 PM	SPLAT File	32,079 KB

Now the serious coroutine!

```
TextureGenerator TextureCoroutine()
{
    std::vector<unsigned int> textureData;

    // Here we convert from a .splat file buffer into a texture
    // With a little bit more foresight perhaps this texture files should have been the native format as it'd be very easy to load it into WebGL
    // a splat is:
    // 6*4 + 4 + 4 = 32
    // XYZ - Position (Float32)
    // XYZ - Scale (Float32)
    // RGBA - colors (uint8)
    // IJKL - quaternion/rot (uint8)

    std::array<float, 4> rot;
    std::array<float, 9> M;
    std::array<float, 6> sigma;
    int texwidth = 2048;
    int texheight;

    std::unique_ptr<SplatData> m_data = co_await std::unique_ptr<SplatData> {};

    int splatCount = m_data->m_ucharBuffer.size() / rowLength;

    texheight = std::ceil((2 * splatCount) / texwidth); // Set to your desired height
    textureData.resize(texwidth * texheight * 4 + 1);

    while (true) {

        for (unsigned int i :   std::views::iota(0u, m_data->m_floatBuffer.size()) |
                                std::views::stride(8) ) {
            // x, y, z
            std::ranges::copy_n(m_data->m_uintBuffer.begin() + i, 3, textureData.begin() + i);

            // r, g, b, a
            std::memcpy(& textureData[i + 7], &m_data->m_ucharBuffer[4 * i + 24], 4);

            // quaternions
            rot[0] = std::bit_cast<float>(m_data->m_ucharBuffer[4 * i + 28 + 0] - 128.0f) / 128.0f;
            rot[1] = std::bit_cast<float>(m_data->m_ucharBuffer[4 * i + 28 + 1] - 128.0f) / 128.0f;
            rot[2] = std::bit_cast<float>(m_data->m_ucharBuffer[4 * i + 28 + 2] - 128.0f) / 128.0f;
            rot[3] = std::bit_cast<float>(m_data->m_ucharBuffer[4 * i + 28 + 3] - 128.0f) / 128.0f;

            // Compute the matrix product of S and R (M = S * R)
            M[0] = m_data->m_floatBuffer[i + 3 + 0] * (1.0f - 2.0f * (rot[2] * rot[2] + rot[3] * rot[3]));
            M[1] = m_data->m_floatBuffer[i + 3 + 1] * (2.0f * (rot[1] * rot[2] + rot[0] * rot[3]));
            M[2] = m_data->m_floatBuffer[i + 3 + 2] * (2.0f * (rot[1] * rot[3] - rot[0] * rot[2]));
            M[3] = m_data->m_floatBuffer[i + 3 + 0] * (2.0f * (rot[1] * rot[2] - rot[0] * rot[3]));
            M[4] = m_data->m_floatBuffer[i + 3 + 1] * (1.0f - 2.0f * (rot[1] * rot[1] + rot[3] * rot[3]));
            M[5] = m_data->m_floatBuffer[i + 3 + 2] * (2.0f * (rot[2] * rot[3] + rot[0] * rot[1]));
            M[6] = m_data->m_floatBuffer[i + 3 + 0] * (2.0f * (rot[1] * rot[3] + rot[0] * rot[2]));
            M[7] = m_data->m_floatBuffer[i + 3 + 1] * (2.0f * (rot[2] * rot[3] - rot[0] * rot[1]));
            M[8] = m_data->m_floatBuffer[i + 3 + 2] * (1.0f - 2.0f * (rot[1] * rot[1] + rot[2] * rot[2]));

            sigma[0] = M[0] * M[0] + M[3] * M[3] + M[6] * M[6];
            sigma[1] = M[0] * M[1] + M[3] * M[4] + M[6] * M[7];
            sigma[2] = M[0] * M[2] + M[3] * M[5] + M[6] * M[8];
            sigma[3] = M[1] * M[1] + M[4] * M[4] + M[7] * M[7];
            sigma[4] = M[1] * M[2] + M[4] * M[5] + M[7] * M[8];
            sigma[5] = M[2] * M[2] + M[5] * M[5] + M[8] * M[8];

            textureData[i + 4] = packHalf2x16(4 * sigma[0], 4 * sigma[1]);
            textureData[i + 5] = packHalf2x16(4 * sigma[2], 4 * sigma[3]);
            textureData[i + 6] = packHalf2x16(4 * sigma[4], 4 * sigma[5]);
        }

        co_yield textureData;
    }
}
```

The texture generation!

```
import splat.coroutine;

TextureGenerator TextureCoroutine()
{
   std::vector<unsigned int> texdata;
   auto m_data = co_await std::unique_ptr<SplatData> {};
   while (true) {
         // for every splats
            // Copy position and color
            // Compute the matrix product of S and R (M = S * R)
            // Set in texture data
         co_yield texdata;
   }
}
```

The texture generator!

```
export struct TextureGenerator {

   struct promise_type {
      //what the coroutine produces
      std::vector<unsigned int> textureData {};

      //what it needs to produce
      std::unique_ptr<SplatData> m_splatData;
```

The texture generator!

```
export struct TextureGenerator {

   struct promise_type {
      std::suspend_always
      yield_value(std::vector<unsigned int> data) {
         textureData = std::move(data);
         return {};
      }
      auto await_transform(std::unique_ptr<SplatData>) {
         struct awaiter {
            std::unique_ptr<SplatData> await_resume() const {
               return std::move(pt.m_splatData); } };
         return awaiter { *this }; }
   };
```

The texture generator!

```
export struct TextureGenerator {
   std::coroutine_handle<promise_type> co_handle {};

   void setData(std::unique_ptr<SplatData> data) {
      co_handle.promise().m_splatData = std::move(data);
   }
   std::vector<unsigned int> texture() {
      return co_handle.promise().textureData;
   }
   void generateTexture() {
      if (co_handle.done())
         return;
      co_handle.resume();
   } };
```

The game is on!

```
// splatty.ixx
import splat.coroutine;

export struct Splatty {
   TextureGenerator textureCoro = TextureCoroutine();

   void setView(const std::array<float, 16>& newviewProj) {
      viewProj = newviewProj;
      if (std::abs(dot - 1) > 0.01) {
         textureCoro.generateTexture();
         texdata = textureCoro.texture();
         m_gl->setTextureData(texdata, texwidth, texheight);

         sortByDepth(viewProj[x], viewProj[y], viewProj[z]);
```

Woohoo!!

But during a late night debugging session...

But during a late night debugging session...

```
export struct TextureGenerator {

   void generateTexture() {
      if (co_handle.done())
         return;
      co_handle.resume();
   }

};
```

But during a late night debugging session...

```
export struct TextureGenerator {

   void generateTexture() {
      if (co_handle.done())
         return;
      std::jthread t([this] {co_handle.resume(); });
   }

};
```

But during a late night debugging session...

But during a late night debugging session...

```
export struct TextureGenerator {

   void generateTexture() {
      if (co_handle.done())
         return;
      co_handle.resume();
   }

};
```

But during a late night debugging session...

```
export struct TextureGenerator {
   std::thread myThread;
   void generateTexture() {
      if (co_handle.done())
         return;
      co_handle.resume();
   }

};
```

But during a late night debugging session...

```
export struct TextureGenerator {
   std::thread myThread;
   void generateTexture() {
      if (co_handle.done() or myThread.joinable())
         return;
      myThread = std::move(std::thread([this] {
         co_handle.resume();
      }));
   }
};
```

Eureka!

Before:

After:

Eureka moments

- Rewriting a JS / WebGL tutorial to C++ / OpenGL
- Surprise! std::embed!!
- Converting fors to ranges/algos
- Native mdspan!!
- Coroutines!
- Threading + coroutines!!!!!!

Recap of the code one last time

https://github.com/joblobob/splatty

```cpp
// main.cpp

#include <QGuiApplication>
#include "GlWindow-splat.h"

int main()
{
    QGuiApplication app(1, "Splatty");

    GLWindowSplat w { std::filesystem::path { "plush.splat" } };
    w.showMaximized();

    return app.exec();
}
```

```cpp
// Glwindow-splat.h

#ifndef GLWIDOWSPLAT_H
#define GLWIDOWSPLAT_H

#include <QOpenGLWindow>
#include <mdspan>

import splatty;

class GLWindowSplat : public QOpenGLWindow
{
    Q_OBJECT

public:
    GLWindowSplat(const std::filesystem::path& splatFilePath) : m_splatty(splatFilePath) {}

    void worldInteraction(std::mdspan<float, std::extents<std::size_t, 4, 4> > view);

private:
    Splatty m_splatty;

    void initializeGL() { m_splatty.initializeGL(); }
    void resizeGL(int w, int h) { m_splatty.resizeGL(w, h); }
    void paintGL();
};

#endif
```

```cpp
// Glwindow-splat.cpp

void GLWindowSplat::paintGL()
{
    // set the view to the new coordinates
    worldInteraction(viewMatrix);
    m_splatty.setView(          viewMatrix[std::array { 0, 2 }],
                                viewMatrix[std::array { 1, 2 }],
                                viewMatrix[std::array { 2, 2 }]);
    //update canvas when we need to ^_^
    update();
}

void GLWindowSplat::worldInteraction(std::mdspan<float, std::extents<std::size_t, 4, 4> > view)
{
    invertMatrix(view);

    rotateMatrix(view, 0.002f, 0.0f, -0.04f, 0.0f);
    translateMatrix(view, 0.01, 0.01, -0.1);

    invertMatrix(view);
}
```

Recap of the code one last time

https://github.com/joblobob/splatty

```
// splatdata.ixx
module;

#include <ranges>
#include <vector>

export module splat.data;

import splat.math;

// 6*4 + 4 + 4 = 8*4
// XYZ - Position (Float32)
// XYZ - Scale (Float32)
// RGBA - colors (uint8)
// IJKL - quaternion/rot (uint8)

export struct SplatData {
    SplatData(const std::vector<unsigned char> splatBuffer) : m_ucharBuffer(splatBuffer),
m_floatBuffer(m_ucharBuffer.size() / 4)
    {
        //copy binary to float with our friend memcpy!
        std::memcpy(m_floatBuffer.data(), m_ucharBuffer.data(), m_ucharBuffer.size());

        //convert float to uint using ranges!
        m_uintBuffer = m_floatBuffer | std::views::transform(to_uints)
                                     | std::ranges::to<std::vector<unsigned int> >();
    }

    std::vector<unsigned char> m_ucharBuffer;
    std::vector<float> m_floatBuffer;
    std::vector<unsigned int> m_uintBuffer;
};
```

```
// splatreader.ixx
module;

#include <vector>

#include <filesystem>
#include <fstream>

export module splat.reader;

export namespace Splat {

std::vector<unsigned char> readFromFile(const std::filesystem::path& path)
{
    // file buffer
    std::vector<unsigned char> u_buffer(std::filesystem::file_size(path));

    // read file data to buffer
    std::basic_ifstream<unsigned char> inputFile(path, std::ios_base::binary);
    inputFile.read(u_buffer.data(), u_buffer.size());
    inputFile.close();

    return u_buffer;
}
}; // namespace Splat
```

```
// splatshaders.ixx

module;

#include <battery/embed.hpp>

export module splat.shaders;

//Please Put My Data In My Executable And Stop Doing Crazy Nonsense, The Feature

export struct ShaderSource {
    static inline const auto vertex   = b::embed<"res/vertexShader.vert">().data();
    static inline const auto fragment = b::embed<"res/fragmentShader.frag">().data();

}; // namespace ShaderSource
```

Recap of the code one last time

```cpp
// splatcoroutine.ixx
module;

#include <coroutine>
#include <expected>
#include <future>
#include <memory>
#include <string>
#include <thread>
#include <vector>

export module splat.coroutine;

import splat.data;

//coroutine!
export struct CountLogger {
    struct promise_type {
        std::string logMessage {};

        CountLogger get_return_object() noexcept { return CountLogger { this }; } // #C Coroutine creation
        std::suspend_never initial_suspend() noexcept { return {}; }              // #D Startup
        std::suspend_always final_suspend() noexcept { return {}; }               // #E Ending
        std::suspend_always yield_value(std::string msg) noexcept {               // #F Value from co_yield
            logMessage = std::move(msg);
            return {};
        }
        void return_value(std::string msg) noexcept { logMessage = std::move(msg); } // #I Value from co_return
        void unhandled_exception() noexcept {}
    };

    std::coroutine_handle<promise_type> co_handle {};

    explicit CountLogger(promise_type* p) : co_handle { std::coroutine_handle<promise_type>::from_promise(*p) } {} // #C Get the handle form the promise
    CountLogger(CountLogger&& rhs) noexcept : co_handle { std::exchange(rhs.co_handle, nullptr) } {}              // #D Move only!

    ~CountLogger() noexcept {
        if (co_handle) {
            co_handle.destroy();
        }
    }

    void count() { // activate it.
        if (not co_handle.done()) {
            co_handle.resume();
        }
    }

    std::string message() { // #F Activate the coroutine and return the data
        if (not co_handle.done()) {
            co_handle.resume();
        }
        return std::move(co_handle.promise().logMessage);
    }
};

//coroutine 2!
export struct TextureGenerator {
    std::thread myThread;

    struct promise_type {
        std::vector<unsigned int> textureData {}; //what the coroutine produces
        std::unique_ptr<SplatData> m_splatData;   //what it needs to produce

        TextureGenerator get_return_object() noexcept { return TextureGenerator { this }; } // #C Coroutine creation
        std::suspend_always initial_suspend() noexcept { return {}; }                        // #D Startup
        std::suspend_always final_suspend() noexcept { return {}; }                          // #E Ending
        std::suspend_always yield_value(std::vector<unsigned int> data) noexcept {           // #F Value from co_yield
            textureData = std::move(data);
            return {};
        }
        void unhandled_exception() noexcept {}
        void return_void() noexcept {}

        auto await_transform(std::unique_ptr<SplatData>) noexcept{ //G Value from co_await 14
                struct awaiter {
                // Customized version instead of using suspend_always or suspend_never
                promise_type& pt;
                constexpr bool await_ready() const noexcept { return true; }
                std::unique_ptr<SplatData> await_resume() const noexcept { return std::move(pt.m_splatData); }
                void await_suspend(std::coroutine_handle<>) const noexcept {}
            };
            return awaiter { *this };
        }
    };

    std::coroutine_handle<promise_type> co_handle {};

    explicit TextureGenerator(promise_type* p) :
        myThread(), co_handle { std::coroutine_handle<promise_type>::from_promise(*p) } {}
    TextureGenerator(TextureGenerator&& rhs) noexcept : co_handle { std::exchange(rhs.co_handle, nullptr) } {} // #D Move only!

    ~TextureGenerator() noexcept { // #E Care taking, destroying the handle if needed
        myThread.join();
        if (co_handle) {
            co_handle.destroy();
        }
    }

    void setData(std::unique_ptr<SplatData> data) { co_handle.promise().m_splatData = std::move(data); }

    void generateTexture() { // #F Activate the coroutine and
        if (!myThread.joinable()) {
            myThread = std::move(std::thread(([this] {
                if (not co_handle.done()) {
                    co_handle.resume();
                }
            })));
        }
    }
    enum class TextureStatus {
            NotReady,
        NoData
    };
    std::expected<std::vector<unsigned int>, TextureGenerator::TextureStatus> texture() { //  return the data
        if (co_handle.done())
            return std::unexpected(TextureStatus::NoData);

        if (co_handle.promise().textureData.size() == 0)
            return std::unexpected(TextureStatus::NotReady);

        return std::move(co_handle.promise().textureData);
    }
};
```

Recap of the code one last time

```
// splatopengl.ixx
module;

#include <QOpenGLBuffer>
#include <QOpenGLContext>
#include <QOpenGLExtraFunctions>
#include <QOpenGLShaderProgram>
#include <QOpenGLTexture>
#include <QOpenGLVertexArrayObject>

#include <mdspan>

export module splat.opengl;

import splat.math;
import splat.shaders;

static std::array<float, 16> defaultViewMatrix = { 0.47, 0.04, 0.88, 0, -0.11, 0.99, 0.02, 0, -0.88, -0.11, 0.47, 0, 0.07, 0.03, 6.55, 1 };

export std::mdspan<float, std::extents<std::size_t, 4, 4> > viewMatrix(defaultViewMatrix.data(), 4, 4);

export constexpr int focalWidth  = 1500;
export constexpr int focalHeight = 1500;

export struct glsplat {
    glsplat(int splatCount) : splatCount(splatCount), m_texture(QOpenGLTexture::Target::Target2D) {}

    QOpenGLTexture m_texture;
    QOpenGLShaderProgram m_program;
    QOpenGLVertexArrayObject m_vao;
    int splatCount = 0;
    int m_projMatrixLoc, m_viewPortLoc, m_focalLoc, m_viewLoc = 0;
    QOpenGLBuffer m_indexBuffer, m_vertexBuffer;
    std::array<float, 16> m_projectionMatrix;

    void viewChanged() {
        QOpenGLExtraFunctions* f = QOpenGLContext::currentContext()->extraFunctions();

        // fps calculations (from paintGL)
        f->glUniformMatrix4fv(m_viewLoc, 1, false, viewMatrix.data_handle());
        f->glClear(GL_COLOR_BUFFER_BIT);
        QOpenGLContext::currentContext()->extraFunctions()->glDrawArraysInstanced(GL_TRIANGLE_FAN, 0, 4, splatCount);
    }

    void initializeGL() {
        QOpenGLExtraFunctions* f = QOpenGLContext::currentContext()->extraFunctions();

        m_program.addShaderFromSourceCode(QOpenGLShader::Vertex, ShaderSource::vertex);
        m_program.addShaderFromSourceCode(QOpenGLShader::Fragment, ShaderSource::fragment);
        m_program.link();
        m_program.bind();

        // Create a VAO. Not strictly required for ES 3, but it is for plain OpenGL.
        if (m_vao.create())
            m_vao.bind();

        f->glDisable(GL_DEPTH_TEST); // Disable depth testing

        f->glEnable(GL_BLEND);
        f->glBlendFuncSeparate(GL_ONE_MINUS_DST_ALPHA, GL_ONE, GL_ONE_MINUS_DST_ALPHA, GL_ONE);

        m_projMatrixLoc  = m_program.uniformLocation("projection");
        m_viewPortLoc    = m_program.uniformLocation("viewport");
        m_focalLoc       = m_program.uniformLocation("focal");
        m_viewLoc        = m_program.uniformLocation("view");

        // positions
        const std::array<float, 8> triangleVertices = { -2, -2, 2, -2, 2, 2, -2, 2 };

        m_vertexBuffer.create();

        f->glBindBuffer(GL_ARRAY_BUFFER, m_vertexBuffer.bufferId());
        f->glBufferData(GL_ARRAY_BUFFER, 8 * 4, triangleVertices.data(), GL_STATIC_DRAW);
        const int a_position = m_program.attributeLocation("position");
        f->glEnableVertexAttribArray(a_position);
        f->glBindBuffer(GL_ARRAY_BUFFER, m_vertexBuffer.bufferId());
        f->glVertexAttribPointer(a_position, 2, GL_FLOAT, GL_FALSE, 0, nullptr);

        f->glBindTexture(GL_TEXTURE_2D, m_texture.textureId());

        auto u_textureLocation = m_program.uniformLocation("u_texture");
        f->glUniform1i(u_textureLocation, 0);

        m_indexBuffer.create();
        const int a_index = m_program.attributeLocation("index");
        f->glEnableVertexAttribArray(a_index);
        f->glBindBuffer(GL_ARRAY_BUFFER, m_indexBuffer.bufferId());
        f->glVertexAttribIPointer(a_index, 1, GL_INT, false, 0);
        f->glVertexAttribDivisor(a_index, 1);
    }

    void resizeGL(int w, int h) {
        QOpenGLExtraFunctions* f = QOpenGLContext::currentContext()->extraFunctions();

        GLfloat tabFloat[] = { focalWidth, focalHeight };
        f->glUniform2fv(m_focalLoc, 1, tabFloat);
        m_projectionMatrix = getProjectionMatrix(focalWidth, focalHeight, w, h);

        GLfloat innerTab[] = { w, h };
        f->glUniform2fv(m_viewPortLoc, 1, innerTab);

        f->glViewport(0, 0, w, h);
        f->glUniformMatrix4fv(m_projMatrixLoc, 1, false, m_projectionMatrix.data());
    }

    void setTextureData(const std::vector<unsigned int>& texdata, int texwidth, int texheight) {
        QOpenGLExtraFunctions* f = QOpenGLContext::currentContext()->extraFunctions();
        f->glBindTexture(GL_TEXTURE_2D, m_texture.textureId());
        f->glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
        f->glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

        f->glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        f->glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

        f->glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32UI, texwidth, texheight, 0, GL_RGBA_INTEGER, GL_UNSIGNED_INT, texdata.data());
        f->glActiveTexture(GL_TEXTURE0);
        f->glBindTexture(GL_TEXTURE_2D, m_texture.textureId());
    }

    void setDepthIndex(const std::vector<unsigned int>& depthIndex) {
        QOpenGLExtraFunctions* f = QOpenGLContext::currentContext()->extraFunctions();

        f->glBindBuffer(GL_ARRAY_BUFFER, m_indexBuffer.bufferId());
        f->glBufferData(GL_ARRAY_BUFFER, depthIndex.size() * 4, depthIndex.data(), GL_DYNAMIC_DRAW);
    }
};
```

Recap of the code one last time

```
// splatmath.ixx
module;

#include <array>
#include <cmath>
#include <mdspan>
#include <print>

#include <bit>
#include <bitset>
export module splat.math;

export std::array<float, 16> getProjectionMatrix(float fx, float fy, int width, int height) {
    constexpr float znear = 0.2f;
    constexpr float zfar  = 200;
    return { (2.0f * fx) / width, 0.f, 0.f, 0.f,
        0.f, -(2 * fy) / height, 0.f, 0.f,
        0.f, 0.f, zfar / (zfar - znear), 1.f,
        0.f, 0.f, -(zfar * znear) / (zfar - znear), 0.f };
}

export void invertMatrix(std::mdspan<float, std::extents<std::size_t, 4, 4> > matrix) {
    float b00 = matrix[std::array { 0, 0 }] * matrix[std::array { 1, 1 }] - matrix[std::array { 0, 1 }] * matrix[std::array { 1, 0 }];
    float b01 = matrix[std::array { 0, 0 }] * matrix[std::array { 1, 2 }] - matrix[std::array { 0, 2 }] * matrix[std::array { 1, 0 }];
    float b02 = matrix[std::array { 0, 0 }] * matrix[std::array { 1, 3 }] - matrix[std::array { 0, 3 }] * matrix[std::array { 1, 0 }];
    float b03 = matrix[std::array { 0, 1 }] * matrix[std::array { 1, 2 }] - matrix[std::array { 0, 2 }] * matrix[std::array { 1, 1 }];
    float b04 = matrix[std::array { 0, 1 }] * matrix[std::array { 1, 3 }] - matrix[std::array { 0, 3 }] * matrix[std::array { 1, 1 }];
    float b05 = matrix[std::array { 0, 2 }] * matrix[std::array { 1, 3 }] - matrix[std::array { 0, 3 }] * matrix[std::array { 1, 2 }];
    float b06 = matrix[std::array { 2, 0 }] * matrix[std::array { 3, 1 }] - matrix[std::array { 2, 1 }] * matrix[std::array { 3, 0 }];
    float b07 = matrix[std::array { 2, 0 }] * matrix[std::array { 3, 2 }] - matrix[std::array { 2, 2 }] * matrix[std::array { 3, 0 }];
    float b08 = matrix[std::array { 2, 0 }] * matrix[std::array { 3, 3 }] - matrix[std::array { 2, 3 }] * matrix[std::array { 3, 0 }];
    float b09 = matrix[std::array { 2, 1 }] * matrix[std::array { 3, 2 }] - matrix[std::array { 2, 2 }] * matrix[std::array { 3, 1 }];
    float b10 = matrix[std::array { 2, 1 }] * matrix[std::array { 3, 3 }] - matrix[std::array { 2, 3 }] * matrix[std::array { 3, 1 }];
    float b11 = matrix[std::array { 2, 2 }] * matrix[std::array { 3, 3 }] - matrix[std::array { 2, 3 }] * matrix[std::array { 3, 2 }];
    float det = b00 * b11 - b01 * b10 + b02 * b09 + b03 * b08 - b04 * b07 + b05 * b06;
    if (det < 0.0000001) {
        return;
    }

    std::array<float, 16> arr { (matrix[std::array { 1, 1 }] * b11 - matrix[std::array { 1, 2 }] * b10 + matrix[std::array { 1, 3 }] * b09) / det,
        (matrix[std::array { 0, 2 }] * b10 - matrix[std::array { 0, 1 }] * b11 - matrix[std::array { 0, 3 }] * b09) / det,
        (matrix[std::array { 3, 1 }] * b05 - matrix[std::array { 3, 2 }] * b04 + matrix[std::array { 3, 3 }] * b03) / det,
        (matrix[std::array { 2, 2 }] * b04 - matrix[std::array { 2, 1 }] * b05 - matrix[std::array { 2, 3 }] * b03) / det,
        (matrix[std::array { 1, 2 }] * b08 - matrix[std::array { 1, 0 }] * b11 - matrix[std::array { 1, 3 }] * b07) / det,
        (matrix[std::array { 0, 0 }] * b11 - matrix[std::array { 0, 2 }] * b08 + matrix[std::array { 0, 3 }] * b07) / det,
        (matrix[std::array { 3, 2 }] * b02 - matrix[std::array { 3, 0 }] * b05 - matrix[std::array { 3, 3 }] * b01) / det,
        (matrix[std::array { 2, 0 }] * b05 - matrix[std::array { 2, 2 }] * b02 + matrix[std::array { 2, 3 }] * b01) / det,
        (matrix[std::array { 1, 0 }] * b10 - matrix[std::array { 1, 1 }] * b08 + matrix[std::array { 1, 3 }] * b06) / det,
        (matrix[std::array { 0, 1 }] * b08 - matrix[std::array { 0, 0 }] * b10 - matrix[std::array { 0, 3 }] * b06) / det,
        (matrix[std::array { 3, 0 }] * b04 - matrix[std::array { 3, 1 }] * b02 + matrix[std::array { 3, 3 }] * b00) / det,
        (matrix[std::array { 2, 1 }] * b02 - matrix[std::array { 2, 0 }] * b04 - matrix[std::array { 2, 3 }] * b00) / det,
        (matrix[std::array { 1, 1 }] * b07 - matrix[std::array { 1, 0 }] * b09 - matrix[std::array { 1, 2 }] * b06) / det,
        (matrix[std::array { 0, 0 }] * b09 - matrix[std::array { 0, 1 }] * b07 + matrix[std::array { 0, 2 }] * b06) / det,
        (matrix[std::array { 3, 1 }] * b01 - matrix[std::array { 3, 0 }] * b03 - matrix[std::array { 3, 2 }] * b00) / det,
        (matrix[std::array { 2, 0 }] * b03 - matrix[std::array { 2, 1 }] * b01 + matrix[std::array { 2, 2 }] * b00) / det };

    std::memcpy(matrix.data_handle(), arr.data(), sizeof(float) * 16);
}

export void rotateMatrix(std::mdspan<float, std::extents<std::size_t, 4, 4> > matrix, float rad, float x, float y, float z) {
    float len = std::hypot(x, y, z);
    x /= len;
    y /= len;
    z /= len;
    float s   = std::sin(rad);
    float c   = std::cos(rad);
    float t   = 1 - c;
    float b00 = x * x * t + c;
    float b01 = y * x * t + z * s;
    float b02 = z * x * t - y * s;
    float b10 = x * y * t - z * s;
    float b11 = y * y * t + c;
    float b12 = z * y * t + x * s;
    float b20 = x * z * t + y * s;
    float b21 = y * z * t - x * s;
    float b22 = z * z * t + c;
    matrix[std::array { 0, 0 }] = matrix[std::array { 0, 0 }] * b00 + matrix[std::array { 1, 0 }] * b01 + matrix[std::array { 2, 0 }] * b02;
    matrix[std::array { 0, 1 }] = matrix[std::array { 0, 1 }] * b00 + matrix[std::array { 1, 1 }] * b01 + matrix[std::array { 2, 1 }] * b02;
    matrix[std::array { 0, 2 }] = matrix[std::array { 0, 2 }] * b00 + matrix[std::array { 1, 2 }] * b01 + matrix[std::array { 2, 2 }] * b02;
    matrix[std::array { 0, 3 }] = matrix[std::array { 0, 3 }] * b00 + matrix[std::array { 1, 3 }] * b01 + matrix[std::array { 2, 3 }] * b02;
    matrix[std::array { 1, 0 }] = matrix[std::array { 0, 0 }] * b10 + matrix[std::array { 1, 0 }] * b11 + matrix[std::array { 2, 0 }] * b12;
    matrix[std::array { 1, 1 }] = matrix[std::array { 0, 1 }] * b10 + matrix[std::array { 1, 1 }] * b11 + matrix[std::array { 2, 1 }] * b12;
    matrix[std::array { 1, 2 }] = matrix[std::array { 0, 2 }] * b10 + matrix[std::array { 1, 2 }] * b11 + matrix[std::array { 2, 2 }] * b12;
    matrix[std::array { 1, 3 }] = matrix[std::array { 0, 3 }] * b10 + matrix[std::array { 1, 3 }] * b11 + matrix[std::array { 2, 3 }] * b12;
    matrix[std::array { 2, 0 }] = matrix[std::array { 0, 0 }] * b20 + matrix[std::array { 1, 0 }] * b21 + matrix[std::array { 2, 0 }] * b22;
    matrix[std::array { 2, 1 }] = matrix[std::array { 0, 1 }] * b20 + matrix[std::array { 1, 1 }] * b21 + matrix[std::array { 2, 1 }] * b22;
    matrix[std::array { 2, 2 }] = matrix[std::array { 0, 2 }] * b20 + matrix[std::array { 1, 2 }] * b21 + matrix[std::array { 2, 2 }] * b22;
    matrix[std::array { 2, 3 }] = matrix[std::array { 0, 3 }] * b20 + matrix[std::array { 1, 3 }] * b21 + matrix[std::array { 2, 3 }] * b22;
}

export void translateMatrix(std::mdspan<float, std::extents<std::size_t, 4, 4> > matrix, float x, float y, float z) {
    matrix[std::array { 3, 0 }] += matrix[std::array { 0, 0 }] * x + matrix[std::array { 1, 0 }] * y + matrix[std::array { 2, 0 }] * z;
    matrix[std::array { 3, 1 }] += matrix[std::array { 0, 1 }] * x + matrix[std::array { 1, 1 }] * y + matrix[std::array { 2, 1 }] * z;
    matrix[std::array { 3, 2 }] += matrix[std::array { 0, 2 }] * x + matrix[std::array { 1, 2 }] * y + matrix[std::array { 2, 2 }] * z;
    matrix[std::array { 3, 3 }] += matrix[std::array { 0, 3 }] * x + matrix[std::array { 1, 3 }] * y + matrix[std::array { 2, 3 }] * z;
}

export int floatToHalf(float val) {
    unsigned int f;
    memcpy(&f, &val, 4);

    int sign = (f >> 31) & 0x0001;
    int exp  = (f >> 23) & 0x00ff;
    int frac = f & 0x007fffff;
    int newExp = 0;
    if (exp < 113) {
        newExp = 0;
        frac |= 0x00800000;
        frac = frac >> (113 - exp);
        if (frac & 0x01000000) {
            newExp = 1;
            frac   = 0;
        }
    } else if (exp < 142) {
        newExp = exp - 112;
    } else {
        newExp = 31;
        frac   = 0;
    }
    return (sign << 15) | (newExp << 10) | (frac >> 13);
}

export unsigned int packHalf2x16(float x, float y) { return std::bitset<32>(floatToHalf(x) | floatToHalf(y) << 16).to_ulong(); }

export constexpr unsigned int to_uints(float v) { return std::bit_cast<unsigned int>(v); }
```

Recap of the code one last time

```
// splatty.ixx - part 1
module;

#include <array>
#include <mdspan>
#include <print>
#include <ranges>
#include <vector>
#include <chrono>
#include <coroutine>
#include <filesystem>
#include <expected>

export module splatty;

import splat.coroutine;
import splat.data;
import splat.opengl;
import splat.math;
import splat.reader;

constexpr int rowLength = 3 * 4 + 3 * 4 + 4 + 4;

CountLogger LoggingCoroutine() // #A Wrapper type Chat containing the promise type
{
    std::println("{}", "Hello! I'm a counting logger");
    std::chrono::steady_clock::time_point begin, end;
    int i = 0;
    begin = std::chrono::steady_clock::now();
    co_yield "Initialization done!";
    while (i < 100) {
        co_yield "Count: " + std::to_string(i++) + " " +
            std::to_string(std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::steady_clock::now() - begin).count()) + " ms";
    }
    co_return "Finished !";
}

TextureGenerator TextureCoroutine()
{
    std::vector<unsigned int> texdata;

    // Here we convert from a .splat file buffer into a texture
    // With a little bit more foresight perhaps this texture file
    // should have been the native format as it'd be very easy to
    // load it into webgl.

    std::array<float, 4> rot;
    std::array<float, 9> M;
    std::array<float, 6> sigma;
    int texwidth = 2048;
    int texheight;
    std::unique_ptr<SplatData> m_data = co_await std::unique_ptr<SplatData> {};

    int splatCount = (m_data->m_ucharBuffer.size() / rowLength);

    texheight = std::ceil((float)(2 * splatCount) / (float)texwidth); // Set to your desired height
    texdata.resize(texwidth * texheight * 4 + 1);

    while (true)
    {
        for (unsigned int i : std::views::iota(0u, m_data->m_floatBuffer.size()) | std::views::stride(8))
        {
            // x, y, z
            std::ranges::copy_n(m_data->m_uintBuffer.begin() + i, 3, texdata.begin() + i);

            // r, g, b, a
            std::memcpy(&texdata[i + 7], &m_data->m_ucharBuffer[4 * i + 24], 4);

            // quaternions
            rot[0] = std::bit_cast<float>(m_data->m_ucharBuffer[4 * i + 28 + 0]) - 128.0f) / 128.0f;
            rot[1] = std::bit_cast<float>(m_data->m_ucharBuffer[4 * i + 28 + 1]) - 128.0f) / 128.0f;
            rot[2] = std::bit_cast<float>(m_data->m_ucharBuffer[4 * i + 28 + 2]) - 128.0f) / 128.0f;
            rot[3] = std::bit_cast<float>(m_data->m_ucharBuffer[4 * i + 28 + 3]) - 128.0f) / 128.0f;

            // Compute the matrix product of S and R (M = S * R)
            M[0] = m_data->m_floatBuffer[i + 3 + 0] * (1.0f - 2.0f * (rot[2] * rot[2] + rot[3] * rot[3]));
            M[1] = m_data->m_floatBuffer[i + 3 + 1] * (2.0f * (rot[1] * rot[2] + rot[0] * rot[3]));
            M[2] = m_data->m_floatBuffer[i + 3 + 2] * (2.0f * (rot[1] * rot[3] - rot[0] * rot[2]));

            M[3] = m_data->m_floatBuffer[i + 3 + 0] * (2.0f * (rot[1] * rot[2] - rot[0] * rot[3]));
            M[4] = m_data->m_floatBuffer[i + 3 + 1] * (1.0f - 2.0f * (rot[1] * rot[1] + rot[3] * rot[3]));
            M[5] = m_data->m_floatBuffer[i + 3 + 2] * (2.0f * (rot[2] * rot[3] + rot[0] * rot[1]));

            M[6] = m_data->m_floatBuffer[i + 3 + 0] * (2.0f * (rot[1] * rot[3] + rot[0] * rot[2]));
            M[7] = m_data->m_floatBuffer[i + 3 + 1] * (2.0f * (rot[2] * rot[3] - rot[0] * rot[1]));
            M[8] = m_data->m_floatBuffer[i + 3 + 2] * (1.0f - 2.0f * (rot[1] * rot[1] + rot[2] * rot[2]));

            sigma[0] = M[0] * M[0] + M[3] * M[3] + M[6] * M[6];
            sigma[1] = M[0] * M[1] + M[3] * M[4] + M[6] * M[7];
            sigma[2] = M[0] * M[2] + M[3] * M[5] + M[6] * M[8];
            sigma[3] = M[1] * M[1] + M[4] * M[4] + M[7] * M[7];
            sigma[4] = M[1] * M[2] + M[4] * M[5] + M[7] * M[8];
            sigma[5] = M[2] * M[2] + M[5] * M[5] + M[8] * M[8];

            texdata[i + 4] = packHalf2x16(4 * sigma[0], 4 * sigma[1]);
            texdata[i + 5] = packHalf2x16(4 * sigma[2], 4 * sigma[3]);
            texdata[i + 6] = packHalf2x16(4 * sigma[4], 4 * sigma[5]);
        }
        co_yield texdata;
    }
}
```

Recap of the code one last time

```
// splatty.ixx - part 2

export struct Splatty
{
    std::unique_ptr<SplatData> m_data;
    std::unique_ptr<glsplat> m_gl;

    int splatCount = 0;

    float lastProjX, lastProjY, lastProjZ;

    static constexpr int texwidth = 2048;
    int texheight;
    std::vector<unsigned int> depthIndex;

    CountLogger log                = LoggingCoroutine(); // #E Creation of the coroutine
    TextureGenerator textureCoro = TextureCoroutine();

    Splatty(const std::filesystem::path& path)
    {
        std::vector<unsigned char> data = Splat::readFromFile(path);

        // resize data following the splatCount
        splatCount = (data.size() / rowLength);

        depthIndex.resize(splatCount + 1);

        texheight = std::ceil((float)(2 * splatCount) / (float)texwidth); // Set to your desired height
        m_data    = std::make_unique<SplatData>(data);
        m_gl      = std::make_unique<glsplat>(splatCount);

        textureCoro.setData(std::make_unique<SplatData>(data));
        textureCoro.generateTexture();
    }

    void sortByDepth(float x, float y, float z)
    {
        int maxDepth = std::numeric_limits<int>::min();
        int minDepth = std::numeric_limits<int>::max();
        std::vector<unsigned int> sizeList(splatCount);
        for (int i = 0; i < splatCount; i++) {
            int depth   = (x * m_data->m_floatBuffer[8 * i + 0] + y * m_data->m_floatBuffer[8 * i + 1] + z * m_data->m_floatBuffer[8 * i + 2]) * 4096;
            sizeList[i] = depth;

            if (depth > maxDepth)
                maxDepth = depth;
            if (depth < minDepth)
                minDepth = depth;
        }

        constexpr int sizeSort = 256 * 256;
        // This is a 16 bit single-pass counting sort
        float depthInv = (sizeSort) / (maxDepth - minDepth);
        std::vector<int> counts0(sizeSort);

        //normalize depth
        sizeList | std::views::transform([&](unsigned int& val) {
            val = std::floor((val - minDepth) * depthInv);
            counts0[val]++; //count occurrences
            return val;
        }) | std::ranges::to<std::vector>();

        std::vector<int> starts0(sizeSort);
        for (int i = 1; i < sizeSort; i++)
            starts0[i] = starts0[i - 1] + counts0[i - 1];

        for (int i = 0; i < splatCount; i++) {
            depthIndex[starts0[sizeList[i]]++] = i;
        }

        m_gl->setDepthIndex(depthIndex);
    }

    void setView(float x, float y, float z)
    {
        log.count(); // #G Send data into the coroutine

        float dot = lastProjX * x + lastProjY * y + lastProjZ * z;

        std::expected<std::vector<unsigned int>, TextureGenerator::TextureStatus> texdata = textureCoro.texture(); // ask the coroutine to generate new data
        if (texdata.error() != TextureGenerator::TextureStatus::NotReady) {
            m_gl->setTextureData(texdata.value(), texwidth, texheight);
        }

        if (std::abs(dot - 1) > 0.01) {
            std::println("{}", log.message()); // #H Wait for more data from the coroutine "here"

            sortByDepth(x, y, z);

            lastProjX = x;
            lastProjY = y;
            lastProjZ = z;
        }

        m_gl->viewChanged();
    }


    void initializeGL() { m_gl->initializeGL(); }

    void resizeGL(int w, int h) { m_gl->resizeGL(w, h); }
};
```

Wow, what a journey!

This cannot end here!

BBQ time!

ly at capacity right now. Processing your captures will take longer than usual.

capture

BBQ time!

tures

Far tree

Flower inf tree

Goals

Failed
Reconstruction failed.
Please try to capture with more overlap and under brighter lighting.

Mauve

Tree depth

Thank you!!