projekt do předmětu **PGR – Počítačová grafika 2023**

Rasterizace na CPU

řešitel: Jakub Kloub, **xkloub03**

Zadání

- Vytvoření rasterizátoru na CPU
- API podobné OpenGL
- Rasterizování do textury
- Možnost vlastních shaderů
    - Vertex shader
    - Fragment shader

Nejdůležitější dosažené výsledky

1. Jednoduchost použití API, které je podobné OpenGL

```
VertexBuffer::Data vbo_data = { …
auto vbo = State::CreateObject(VertexBuffer(std::move(vbo_data)));
auto ibo = State::CreateObject(IndexBuffer({     // Cube
  0, 1, 2, 2, 1, 3,       // Near
  1, 5, 3, 3, 5, 7,       // Right
  5, 4, 7, 7, 4, 6,       // Far
  4, 0, 6, 6, 0, 2,       // Left
  4, 1, 0, 4, 5, 1,       // Top
  6, 2, 3, 6, 3, 7,       // Bottom
}));
vao = State::CreateObject(VertexArray({
  { vbo, AttributeType::Vec3, 6 * sizeof(float), 0 },
  { vbo, AttributeType::Vec3, 6 * sizeof(float), 3 * sizeof(float) },
}, ibo));
fb = State::CreateObject(Framebuffer::CreateBasic(vp_size));

/// Vertex shader ///
auto vs_func = [](VertexShader* vs) {
  // Vertex attributes
  auto aPos = vs->Attribute<glm::vec3>(0).value().get();
  auto aColor = vs->Attribute<glm::vec3>(1).value().get();
  // Uniforms
  auto mvp = vs->Uniform<glm::mat4>("mvp").value().get();
  // Computation
  vs->Out<glm::vec3>("color") = aColor;
  vs->m_Position = mvp * glm::vec4(aPos, 1.0f);
};

/// Fragment shader ///
auto fs_func = [](FragmentShader* fs) {
  auto color = fs->In<glm::vec3>("color");
  fs->m_FragColor = glm::vec4(color, 1.0f);
};

prg = State::CreateObject(Program({
  .vertex_shader = State::CreateObject(VertexShader(vs_func)),
  .fragment_shader = State::CreateObject(FragmentShader(fs_func)),
}));

State::SetDepthTest(true);
State::SetCullFace(CullFace::None);
```

Figure 1 Vytvoření objektů

```
fb->Use();
State::Clear(Colors::Gray);
prg->Use();
prg->SetUniform("mvp", mvp);
vao->Use();
State::DrawIndexed(Primitive::Triangles, vao->GetIndexBuffer()->data.size());
```

Figure 2: Vykreslování

2. Možnost rasterizace do obecné textury bytů. To nám dovoluje použít rasterizér v mnoha různých kontextech. Například bychom mohli zobrazovat texturu pomocí knihovny NCurses v terminálu.
3. Vlastní shadery

Ovládání vytvořeného programu

- Abych měl jak zobrazit výslednou texturu, tak jsem vytvořil program, který používá moje API a vykresluje texturu v Dear ImGui okně.
- Program stačí spustit a (pokud bude potřeba) zvětšit příslušné okno se zobrazenou texturou.

Zvláštní použité znalosti

V rámci projektu jsem nepoužil žádné znalosti navíc.

Použité technologie

- Ke zkompilování projektu je potřeba program Meson, který už stáhne všechny potřebné závislosti a zkompiluje program.
- Program je závislý na několika jiných knihohvnách:
    - **GLM** – Použito pro matematické funkce a datové typy
    - **ImGuiWrapper** – Jedná se o mojí vlastní knihovnu, která mi dovoluje jednoduše sestrojit nový program který u6 má nakonfigurované ImGui a OpenGL prostředí.
    - **Ren-Utils** – Také se jedná o mojí knihovnu. Tato knihovna poskytuje užitečné nástroje, mezi které patří například formátování std::string nebo logovací API.

Použité zdroje

Zde vypište, které zdroje jste použili k tvorbě: hotový kód, hotová data (obrázky, modely, …), studijní materiály. Pokud vyplyne, že v projektu je použit kód nebo data, která nejsou uvedena tady, jedná se o závažný problém a projekt bude pravděpodobně hodnocen 0 body.

Rozsah: potřebný počet odrážek

- Projektu je založen pouze na studijních materiálech z předmětu IZP a PGR společně z dokumentací jednotlivých funkcí OpenGL

Co bylo nejpracnější

Nejpracnější bylo navrhnout hlavní kostru API. Chtěl jsem, aby kostra šla jednoduše rozšiřovat o novou funkcionalitu. Po návrhu kostry bylo obtížné implementovat samotnou rasterizaci. Zde tvořilo potíže efektivní předávání dat bez nutnosti zbytečných kopírování. Dále se tato sekce velice špatně ladí, protože dokud není implementována velká část pipeline, tak program nic nedělá. Právě kvůli ladění jsem nakonec strávil nejvíce častu u implementace rasterizace.

Zkušenosti získané řešením projektu

Popište, co jste se řešením projektu naučili. Zahrňte dovednosti obecně programátorské, věci z oblasti počítačové grafiky, ale i spolupráci v týmu, hospodaření s časem, atd.

Rozsah: formulujte stručně, uchopte cca 3-5 věcí

Projekt mi dokonale osvěžil znalosti OpenGL pipeline, kterou jsem si musel detailně zopakovat. Dále jsem jsem si vyzkoušel strukturu API, která pracuje s velkým globálním stavem. Protože tento stav uchovává instance všech objektů, implementoval jsem tzv. Handle objekty, se kterýma se dá pracovat jako s instancí původního objektu, což nás odstiňuje od přímé interakce s globálním stavem. Tento styl programování jsem dělal poprvé.

Autoevaluace

Ohodnoťte vaše řešení v jednotlivých kategoriích (0 – nic neuděláno, zoufalství, 100% – dokonalost sama). Projekt, který ve finále obdrží plný počet bodů, může mít složky hodnocené i hodně nízko. Uvedení hodnot blízkých 100% ve všech nebo mnoha kategoriích může ukazovat na nepochopení problematiky nebo na snahu kamuflovat slabé stránky projektu. Bodově hodnocena bude i schopnost vnímat silné a slabé stránky svého řešení.

Technický návrh: 60% (analýza, dekompozice problému, volba vhodných prostředků, …)

API není dokonalé a jeho struktura zatím nepodporuje více náročné funkcionality OpenGL

Programování: 68% (kvalita a čitelnost kódu, spolehlivost běhu, obecnost řešení, znovupoužitelnost, …)

S kódem jsem v celku spokojený. Vadí mi někdy nepřesné názvy, nebo lehce matoucí struktura složek. Také chybí spousta kokmentářů.

Vzhled vytvořeného řešení: 40% (uvěřitelnost zobrazení, estetická kvalita, vhled GUI, …)

Projekt není dotažený dokonce. Podporuje pouze triviální rasterizaci trojúhelníků a shadery. Nestihl jsem kvůli tomu udělat interaktivní GUI pro konfiguraci rasterizéru.

Využití zdrojů: 69% (využití existujícího kódu a dat, využití literatury, …)

Zdroje jsem využíval hojně a efektivně. Navazoval hlavně na své zkušenosti z IZG. Na druhou stranu jsem ale nepoužil více zrojů jiných než studijní materiály FITu.

Hospodaření s časem: 25% (rovnoměrné dotažení částí projektu, míra spěchu, chybějící části řešení, …)

Projekt jsem začal dělat až moc pozdě. Pro jeho dotažení bych potřeboval alespoň ještě 2 dny, které jsem bohužel již neměl. Špatně jsem si rozplánoval časový harmonogram.

Celkový dojem: 90% (pracnost, získané dovednosti, užitečnost, volba zadání, cokoliv, …)

Projekt se mi zdá pracný hlavně časově, ale je velmi zajímavý. Při jeho tvorbě jsem se naučil novým technikám C++ a našel nové typy bugů na které musím dávat pozor (např. typ std::unordered_map<const char*, cokoliv> — klíč není řetězec)

Doporučení pro budoucí zadávání projektů

Vše mi vyhovovalo.