make impossible state unrepresentable

Lesley Lai

http://lesleylai.info/

Motivation

```
1  struct QueueFamilyIndices {
2      std::optional<uint32_t> graphics;
3      std::optional<uint32_t> present;
4      std::optional<uint32_t> compute;
5
6      bool isComplete() const {
7          return graphics.has_value()
8          && present.has_value()
9          && compute.has_value();
10     }
11 };
```

Motivation

```
1  QueueFamilyIndices findQueueFamilies(/*...*/) {
2    // ...
3    QueueFamilyIndices indices;
4    for (const auto& queue: queues) {
5      if (/* queue i support graphics */) {
6          indices.graphics = i;
7      }
8
9      if (/* queue i support present */) {
10         indices.present = i;
11     }
12
13     if (/* queue i support compute */) {
14         indices.compute = i;
15     }
16
17     if (indices.isComplete()) {
18         break;
19     }
20   }
21   return indices;
22 }
```

Transformation

```
1 struct QueueFamilyIndices {
2     uint32_t graphics;
3     uint32_t present;
4     uint32_t compute;
5 };
```

Transformation

```
1  std::optional<QueueFamilyIndices> findQueueFamilies(/*...*/) {
2    // ...
3    std::optional<uint32_t> graphicsFamily = std::nullopt;
4    std::optional<uint32_t> presentFamily = std::nullopt;
5    std::optional<uint32_t> computeFamily = std::nullopt;
6
7    for (const auto& queue: queues) {
8      if (/* queue i support graphics */) {
9          graphicsFamily = i;
10     }
11
12     if (/* queue i support present */) {
13         presentFamily = i;
14     }
15
16     if (/* queue i support compute */) {
17         computeFamily = i;
18     }
19
20     if (graphicsFamily && presentFamily && computeFamily) {
21         return QueueFamilyIndices{*graphicsFamily, *presentFamily,
22                                   *computeFamily};
23     }
24   }
25
26   return std::nullopt;
27 }
```

Memory footprint gets reduced

Less assertion or run-time checking

API becomes cleaner

An example with variant

```
1  struct DrawCommand {
2    std::uint32_t count;
3    std::uint32_t vertex_offset;
4    std::uint32_t instance_count;
5  };
6
7  struct DrawIndirectCommand {
8    void* indirect;
9  };
10
11 struct BindGraphicsPipelineCommand {
12   GraphicsPipelineHandle pipeline;
13 };
14
15 using Command = std::variant<DrawCommand, DrawIndirectCommand
16                              BindGraphicsPipelineCommand, ...>;
17
18 struct CommandBuffer {
19   void push_command(Command command);
20   std::vector<Command> commands;
21 };
```

An example with variant

```
1  struct DrawCommand {
2    std::uint32_t count;
3    std::uint32_t vertex_offset;
4    std::uint32_t instance_count;
5
6    GraphicsPipelineHandle pipeline;
7  };
8
9  struct DrawIndirectCommand {
10   void* indirect;
11
12   GraphicsPipelineHandle pipeline;
13 };
14
15 using Command = std::variant<DrawCommand, DrawIndirectCommand>;
16
17 class CommandBuffer {
18   void push_command(Command command);
19   std::vector<Command> commands;
20 };
```

An example with variant

```
1  struct DrawCommand {
2    std::uint32_t count;
3    std::uint32_t vertex_offset;
4    std::uint32_t instance_count;
5  };
6
7  struct DrawIndirectCommand {
8    void* indirect;
9  };
10
11 using Command = std::variant<DrawCommand, DrawIndirectCommand>;
12
13 class SecondaryCommandBuffer {
14   void push_command(Command command);
15   std::vector<Command> commands;
16   GraphicsPipelineHandle pipeline;
17 };
18
19 class CommandBuffer {
20   void push_commands(SecondaryCommandBuffer buffer);
21   std::vector<Command> secondary_buffers;
22 };
```

The pitfall of Move semantics

The pitfall of Move semantics

```
1 class Window {
2   // ...
3
4   Window(Window&& other) noexcept = delete;
5   Window& operator=(Window&& other) noexcept = delete;
6
7 private:
8   std::reference_wrapper<GLFWwindow> window;
9 }
```

The pitfall of Move semantics

```
1  class Window {
2    // ...
3
4    Window(Window&& other) noexcept : window{other.window} {
5      other.window = nullptr;
6    }
7
8  private:
9    GLFWwindow* window;
10 }
```

Thank you