05 Scaling and Alpha Blending

Functional and Implementation Guidelines

Functional Guidelines

1. Your application should implement dynamic scaling of textures (both upscaling and downscaling)
   1. “Scaling” would translate for an image to change it’s dimensions at run-time
   2. apply some scaling experiments in your game in order to demonstrate that scaling is working as expected
2. Your application should implement alpha blending (transparency change) at runtime for individual textures
   1. apply some alpha blending experiments in your game in order to demonstrate that alpha blending is working as expected
3. Try to incorporate the Flyweight design pattern for assets storing/loading
   1. This would mean that if 2 game textures are using the same image - that image should not be loaded twice. It should be loaded only once and reused (shared)
   2. There should be a “central” run-time repository for texture storing/loading

Implementation Guidelines

1. **How scaling works (upscale and downscale) - Lecture**
2. Implement renderer\_hints to enable linear interpolation
   1. Renderer hints must be enabled before the renderer is actually created

| #include <SDL\_hints.h>  if (!SDL\_SetHint(SDL\_HINT\_RENDER\_SCALE\_QUALITY, "1")) {  std::cerr << "Warning: Linear texture filtering not enabled! "  "SDL\_SetHint() failed. SDL Error: " << SDL\_GetError() << std::endl;  return EXIT\_FAILURE;  } |
| --- |

1. Experiment with scaling
   1. (hack it in the Render function)

| SDL\_Rect destRect = …;  SDL\_RenderCopy(renderer->sdlRenderer, texture, NULL, &destRect) |
| --- |

* 1. Then revert it back

1. Flyweight pattern - <https://en.wikipedia.org/wiki/Flyweight_pattern>
2. Implement DrawParams struct
   1. Structure

| struct DrawParams {  //Top left position of texture  Point pos = Point::ZERO;  //Draw dimensions of the texture  int32\_t width = 0;  int32\_t height = 0;  //unique resourceId  int32\_t rsrcId = -1;  }; |
| --- |

* 1. Don’t forget to reset your struct - No variables should ever be uninitialized!
  2. Either give initial values to all params inside the struct or do so in the constructor
  3. Modify Renderer to work accept draw params as well as SDL\_Texture

| void drawImage(const DrawParams& drawParams, SDL\_Texture \*texture); |
| --- |

1. Implement an ImageContainer class
   1. Structure

| class ImageContainer {  //the textures we'll be drawing  std::unordered\_map<int32\_t, SDL\_Texture\*> \_textures;  std::unordered\_map<int32\_t, Rectangle> \_textureFrames;  }; |
| --- |

* 1. Public API

| SDL\_Texture\* getImageTexture(int32\_t rsrcId) const;  Rectangle getImageFrame(int32\_t rsrcId) const; |
| --- |

* 1. ImageContainerCfg

| struct ImageConfig {  std::string location;  int32\_t width = 0;  int32\_t height = 0;  };  struct ImageContainerConfig {  std::unordered\_map<int32\_t, ImageConfig> imageConfigs;  }; |
| --- |

1. EngineConfig should no longer populate the GameConfig
   1. Instead, populate the ImageContainerCfg and let the game reuse it

| struct EngineConfig {  MonitorWindowConfig windowCfg;  ImageContainerConfig imageContainerCfg;  }; |
| --- |

1. Make the new ImageContainer part of the Engine
2. Change the game structure to hold a **const** pointer to the ImageContainer and use it

| class Game {  DrawParams \_gameImg;  DrawParams \_layer2Img;  const ImageContainer \*\_imageContainer = nullptr; //TODO remove me  }; |
| --- |

1. Modify the Game to use KEY\_Q/W/E/R to expand/shrink width/height
2. Modify the Game to use KEY\_UP/DOWN/LEFT/RIGHT to change the position by 10 pixels
3. **Alpha blending - Lecture**
4. Expand DrawParams adding opacity and blendMode

| enum class BlendMode : uint8\_t {  NONE = 0, //value for SDL\_BLENDMODE\_NONE  BLEND = 1, //value for SDL\_BLENDMODE\_BLEND  ADD = 2, //value for SDL\_BLENDMODE\_ADD  MOD = 4 //value for SDL\_BLENDMODE\_MODE  };  inline constexpr int32\_t FULL\_OPACITY = 255;  inline constexpr int32\_t ZERO\_OPACITY = 0; |
| --- |

1. Expand Texture set of functionalities
   1. Changing the blend mode

| int32\_t setBlendModeTexture(SDL\_Texture \*texture, BlendMode blendMode); |
| --- |

* 1. Internal SDL method

| SDL\_SetTextureBlendMode() |
| --- |

* 1. Changing the alpha modulation (opacity)

| int32\_t setAlphaTexture(SDL\_Texture \*texture, int32\_t alpha); |
| --- |

* 1. Internal SDL method

| SDL\_SetTextureAlphaMod() |
| --- |

1. Enable Blending mode BlendMode::BLEND for all textures inside the ImageContainer
   1. Temporary hack that enables opacity on all textures
2. Flyweight design pattern + opacity = trouble
   1. Multiple images are reusing the same texture
   2. Blending is applied to the texture
   3. Opacity is changed on one of them -> it changes for all of them
3. Solution - delay the actual blending prior to drawing
   1. FULL\_OPACITY -> happy case

| if (FULL\_OPACITY == drawParams.opacity) {  SDL\_RenderCopy(\_sdlRenderer, texture, nullptr, &destRect);  } |
| --- |

* 1. Non FULL\_OPACITY -> complex case
  2. change opacity
  3. draw
  4. revert opacity back to FULL\_OPACITY

| if (FULL\_OPACITY == drawParams.opacity) {  SDL\_RenderCopy(\_sdlRenderer, texture, nullptr, &destRect);  } else {  Texture::setAlpha(texture, drawParams.opacity);  SDL\_RenderCopy(\_sdlRenderer, texture, nullptr, &destRect);  Texture::setAlpha(texture, drawParams.opacity)  } |
| --- |

1. Modify the Game to use KEY\_A/S to change the opacity by 20