13 Building our own game (4/4) - Draw Optimizations

Functional and Implementation Guidelines

Functional Guidelines

1. Add additional logic in your graphical debug console (if you have one implemented)
   1. Keep track of how many draw calls your application is making per single frame
   2. Think of a way how this can be changed via a Frame Buffer Object?
2. Implement a Frame Buffer Object(FBO) set of functionalities. The FBO should support:
   1. Using of the existing Image/Text alike API for creation/destruction/drawing
   2. Use the flyweight design pattern
   3. Have a single “repository” that would store all the run-time generated FBO textures
   4. Ability to “clear” the content of the FBO with a user provided color
3. Try to update your application and apply FBO pattern to where it is most needed (almost ‘static’ or purely static textures)
4. Transform the game board “flip” animation by doing an actual rotation animation
5. Get creative - the sky's the limit!

Implementation Guidelines

1. **Draw Optimizations - Lecture**
2. Implement widgets count in the EngineDebugConsole.
   1. Use only a single integer as a counter in the Renderer
   2. How can this be optimized using a Frame Buffer Object?
3. Extend the Renderer/Texture Public API

| int32\_t clearCurrentRendererTarget(const Color &color);  int32\_t setRendererTarget(SDL\_Texture \*target);  void setRenderer(SDL\_Renderer \*renderer);  void resetRendererTarget() |
| --- |

* 1. New Internal SDL Methods

| SDL\_SetRenderTarget()  SDL\_CreateTexture() |
| --- |

* 1. Implementation

| SDL\_Texture\* texture =  SDL\_CreateTexture(\_sdlRenderer, SDL\_PIXELFORMAT\_RGBA8888,  SDL\_TEXTUREACCESS\_TARGET, width, height); |
| --- |

1. Change the flags of the SDL\_Renderer to SDL\_RENDERER\_TARGETTEXTURE
   1. This supports streaming to a texture
2. Expand the Renderer + DrawMgr Public API

| struct Renderer {  SDL\_Renderer \*\_sdlRenderer = nullptr;  Color \_clearColor = Colors::BLACK;  int32\_t \_widgetsCount { 0 };  bool \_isRendererLocked; //allows or forbids changing to different renderer target  };  int32\_t lockRenderer();  int32\_t unlockRenderer();  void resetRendererTarget();  void clearCurrentRendererTarget(const Color &color);  void changeRendererTarget(SDL\_Texture\* texture); //soon to be changed to ‘int32\_t fboId’ |
| --- |

1. Implement FboContainer class
   1. (Copy the TextContainer and remove the unnecessary code)
   2. Structure

| class FboContainer {  //the textures we'll be drawing  std::vector<SDL\_Texture\*> \_textures;  }; |
| --- |

* 1. Public API

| void createFbo(int32\_t fboWidth, int32\_t fboHeight, int32\_t &outFboId);  void unloadFbo(int32\_t fboId);  SDL\_Texture\* getFboTexture(int32\_t fboId) const; |
| --- |

* 1. Add the FboContainer to the RsrcMgr (inherit)

1. Extend the DrawParams
   1. Add new WidgetType::FBO
   2. Add fboId in the union.
2. Expand the Renderer/DrawMgr to account for the new widget type when drawing
   1. Add Renderer+DrawMgr changeRendererTarget(int32\_t fboId)
3. Implement FBO class
   1. (Copy the Text One) and modify as fit
   2. Structure

| class FBO: public Widget {  std::vector<DrawParams> \_storedItems;  Color \_clearColor = Colors::BLACK;  bool \_isDestroyed = false;  bool \_isLocked = true;  }; |
| --- |

* 1. Public API

| void create(int32\_t width, int32\_t height, const Point &pos = Point::ZERO,  const Color &clearColor = Colors::BLACK);  void destroy();  void unlock();  void lock();  void reset();  void addWidget(const Widget &widget);  void update(); |
| --- |

* 1. Usage

| void Game::regenerateGameFbo() {  \_gameFbo.unlock();  \_gameFbo.reset();  //populate the game->gameFbo  \_gameFbo.update();  \_gameFbo.lock();  } |
| --- |

1. Implement Renderer/DrawMgr method updateCurrRenderer target
   1. This bypasses the addDrawCmd/drawWidget command and widget counter
2. Implement an FBO in the PieceHandler that would contain all of the pieces.
3. WHY is the screen black? Because the FBO background is BLACK
   1. Activate alpha modulation and set COLOR\_FULL\_TRANSPARENT after creation
4. WHY when you move a piece - the board does not get updated?
5. Move the FBO from the PieceHandler to the Game.cpp
6. Draw all of PieceHandler widgets on the \_gameFbo

| void draw(FBO &boardFbo); |
| --- |

1. Do the same for the GameBoard

| void drawBoardOnFbo(FBO &gameFbo); |
| --- |

* 1. Warning: Export only the game board image to the FBO, otherwise layering will be lost
  2. keep using the drawBoard() method as before

1. Expand the GameInterface

| virtual void onEndTurnAnimFinished() = 0; |
| --- |

1. Expend the BoardAnimator class
   1. Use a timer to do the actual rotation animation
   2. Change the structure to use a FBO instead of a single Image

| class GameBoardAnimator: public TimerClient {  GameInterface \*\_gameInterface = nullptr;  FBO \*\_boardFbo = nullptr;  int32\_t \_timerId { 0 };  int32\_t \_currRotation { 0 };  int32\_t \_targetRotation { 0 };  WidgetFlip \_targetFlipType = WidgetFlip::NONE;  }; |
| --- |

* 1. Add isActive() method
  2. Add the background image to the FBO depending on that is the animator active or not

1. Activate the WidgetSetFlip on the PieceHandler
2. Get creative!