Google Summer of Code 2019

Name: Kartik Kumar
Institute: National Institute of Technology Hamirpur
Email: kartik.sckorpio@gmail.com
GitHub: sckorpio
Location: Hamirpur, Himachal Pradesh, India(+5:30 UTC)

LibreCAD 3 OpenGL Rendering

NEED?

LibreCAD is a free Open Source CAD application for Windows, Apple and Linux. It allows industrial designers and graphics enthusiast to create CAD projects of the highest standard and precision. With this precision and standard comes the **need** of high quality and accelerated **rendering** to visualise a document. LibreCAD 3 was designed to have multiple rendering engines without major modifications to its core.

Right now LibreCAD 3 uses **Cairo** for rendering. For users with high resolution screens, Cairo is not convenient because of a **slowness**, due to a bad integration with Qt and a **missing caching system**. At each frame when rendering is done the data is sent from CPU to GPU which is very inefficient.

WHY OpenGL ?

OpenGL (Open Graphics Library) is a software interface to graphics hardware. It is the industry's most widely used and supported 2D and 3D graphics application programming interface (API).The interface consists of a number of function calls which can be used to draw complex two and three-dimensional scenes from simple geometric primitives such as points, lines, and polygons.

OpenGL has **lower CPU overhead** for draw calls and state changes and lets you take **advantage of the GPU** to render graphics on your device's screen and performance is excellent. Also OpenGL is **easily integrable with Qt** (platform which LibreCAD uses)for windowing system and event management.

Brief Project Summary

The project consists of replacing Cairo with OpenGL in LibreCAD 3.
My goal is to make a complete well abstracted OpenGL implementation with C++ for the rendering in LibreCAD. The OpenGL rendering engine should be compatible with the painter system and the rendering calls syntax used in drawing entities.

It should use the Buffer Objects in GPU for accelerated performance. There should be a caching mechanism to save the drawn entities (till they are changed) in CPU and the corresponding buffer objects in GPU, so that at each frame there should not be need of sending data from CPU to GPU and this will increase the performance.

It should be able to draw lines, circles, arcs etc .(simple entities) and also be able to render complex entities like dimensions (which are a combination of lines, endcaps , arrows, text). Have features like pan, zoom etc and other transformations. Should have a good and optimized Text rendering support. Some more features like gradient, line patterns etc.

NOTE: This proposal is a continuation of the partial implementation of the painter which is already been done by me and i am currently integrating it with LibreCAD 3

Detailed Project Description & Implementation details

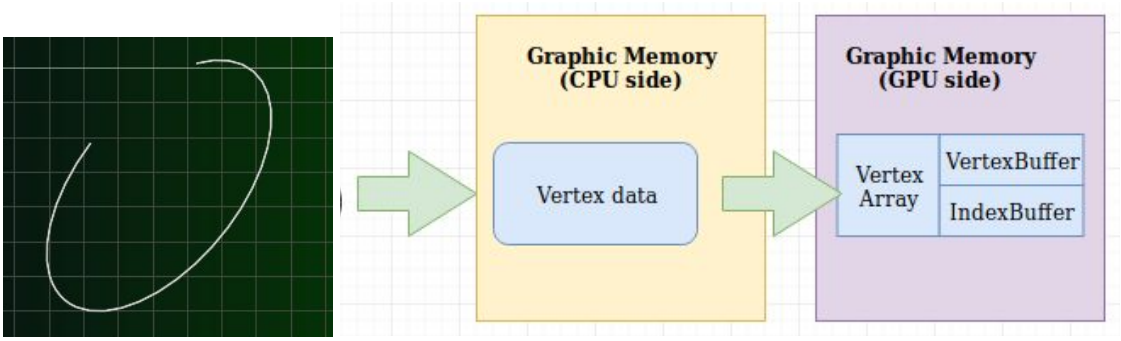
There are various aspects of the project:

1. Using Buffer Objects
2. Complete well abstracted opengl rendering engine
3. Making compatible with the LibreCAD painter system
4. Caching mechanism
5. Pan, Zoom and other transformation facilities
6. Text Rendering
7. Other features like gradient, line patterns etc

1. Using Buffer Objects

There are basically **two** modes of rendering in OpenGL API . First one is the **immediate mode (Legacy)** and the **restrained mode** or the core profile **(Modern one).**The immediate mode is really easy to use and understand, but it is also extremely inefficient. When using OpenGL's core-profile, OpenGL forces us to use modern practices like use of buffer objects, shaders, matrices etc

For example: an arc-ellipse drawn in librecad

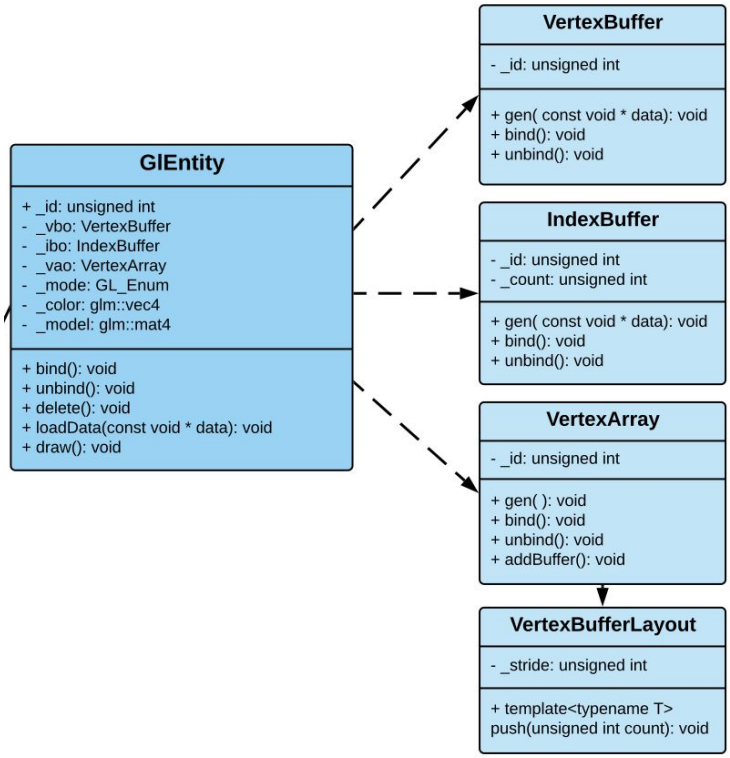


The process has the following steps--

- Create a Vector Buffer Object that will store the vertices on video memory.
- Creating VertexArray Object
- Set up the vertexbuffer layout for it which specifies how the data is stored in the buffer and what they represent.
- Compile the shader and create shader program.
- Connecting the vertexAttribPointer with the Vertex shader and fragment shader.
- Bind the Bufferobjects
- Bind the Shader
- Set Uniforms in shader
- Draw with the required render mode.

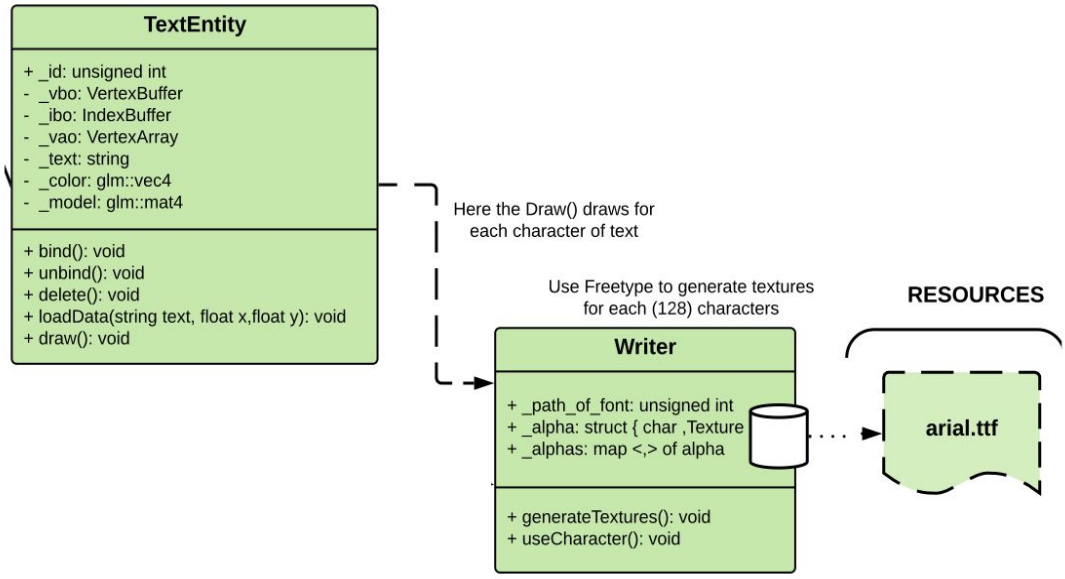
2. Complete well abstracted OpenGL rendering engine

GlEntity : It is the well abstracted class which holds the vertexbuffer, index buffer , vertexbufferlayout, render mode, color, model matrix etc.

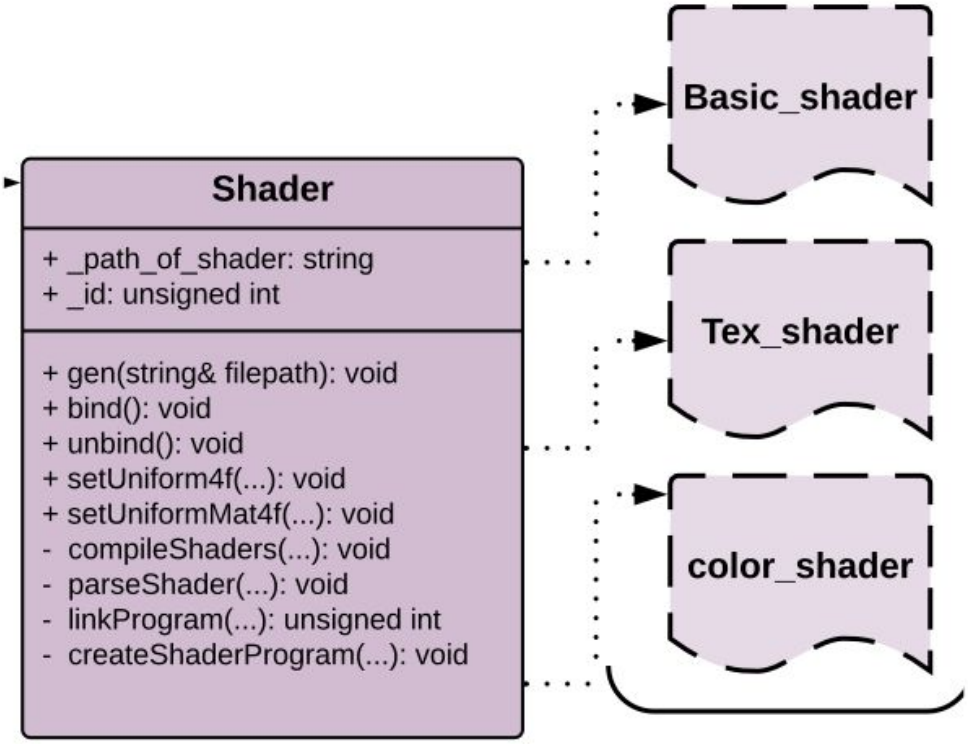


TextEntity : It holds the buffer objects and the text string to be rendered , for drawing it uses the already saved textures for font.

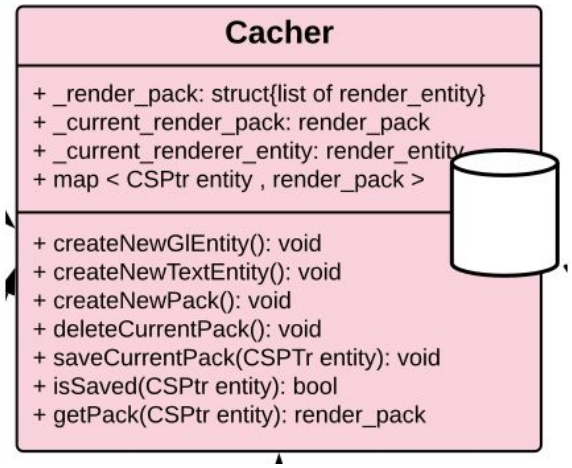
Writer : It is responsible for the creation for textures for characters of font (.ttf) and saving them.



RenderEntity : parent class for GlEntity and TextEntity.
Shader : It is the class for abstracting shader manipulations which loads, compile, setUniforms of the shader written in GLSL.

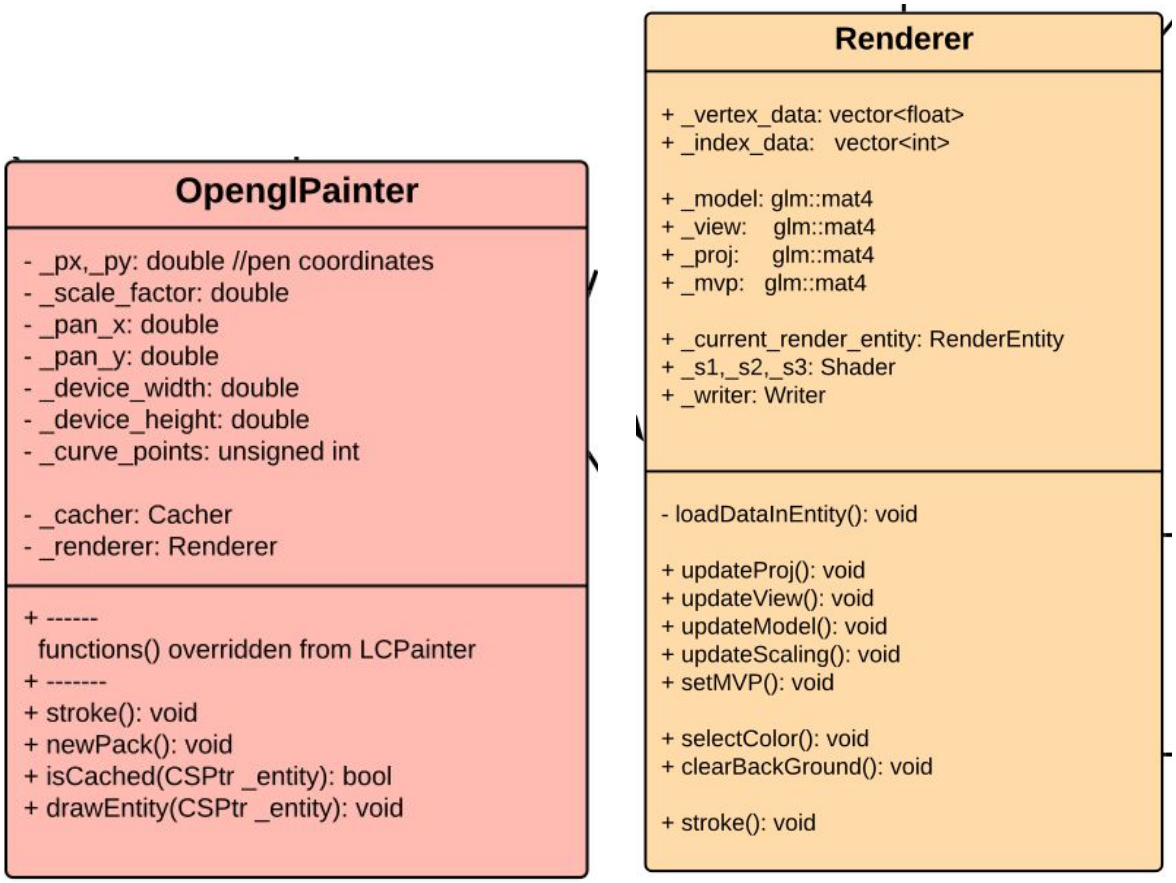


Cacher: It is the place for caching the render_packs(list of render entities) with their corresponding id's and in turn are linked to the GPU buffers.



Caching_painter: This a part of normal painter which itself inherits the LcPainter, the only difference with this is that it does not perform any rendering instead it is responsible for making render_entity, push them in pack and cache them. It sends data to the GPU but doesn't render.

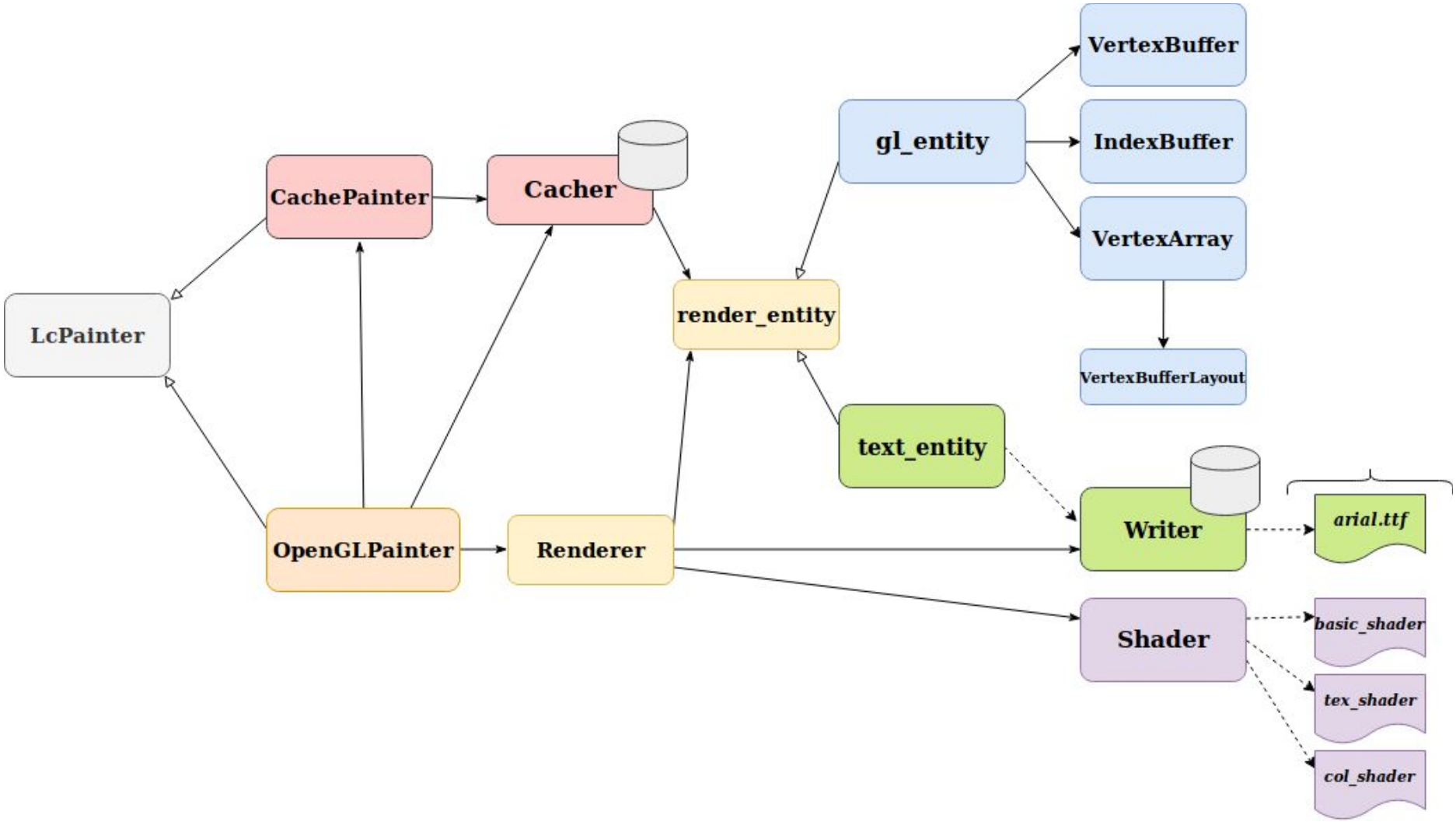
Renderer : This is the one which is responsible for rendering, It is also responsible for the matrix manipulation for pan,zoom etc(manipulating the global MVP matrix). It also sets the uniform of shaders



OpenGLPainter : This the outermost layer of this rendering package,
Here there are no opengl calls, all the inner pure opengl calls are mapped to this painter which enables LibreCAD to use its own painter syntax to render things.

All these classes are related to each other and with the openglpainter outermost abstraction level the opengl rendering package can be used with the painter syntax of LibreCAD.

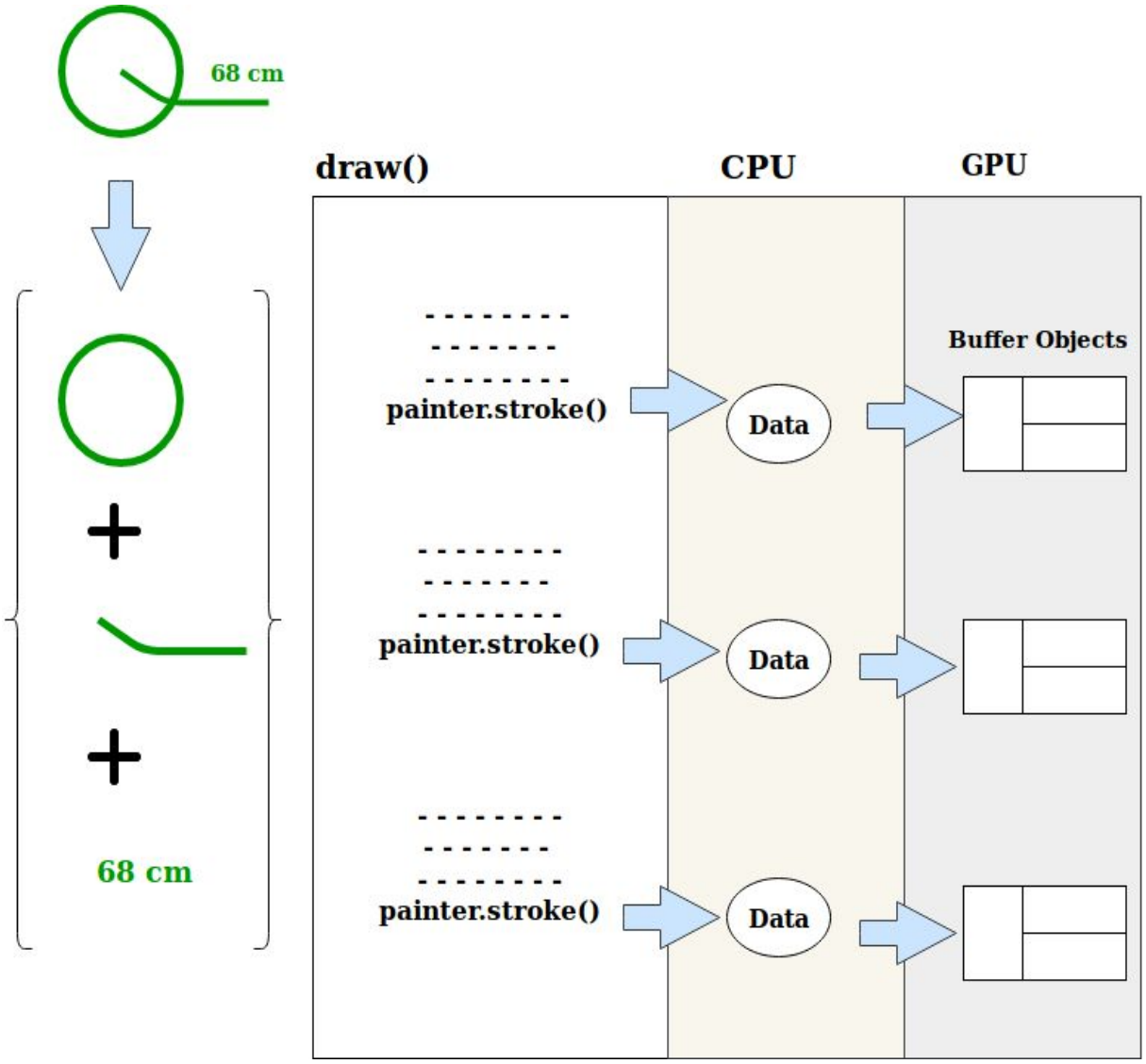
On next page is the complete diagram of how these classes are linked to each other.


VertexBuffer
gl_entity
IndexBuffer
CachePainter
Cacher
VertexArray
render_entity
LcPainter
VertexBufferLayout
text_entity
Writer
arial.ttf
OpenGLPainter
Renderer
Shader
basic_shader
tex_shader
col_shader

3. Compatible with librecad painter system

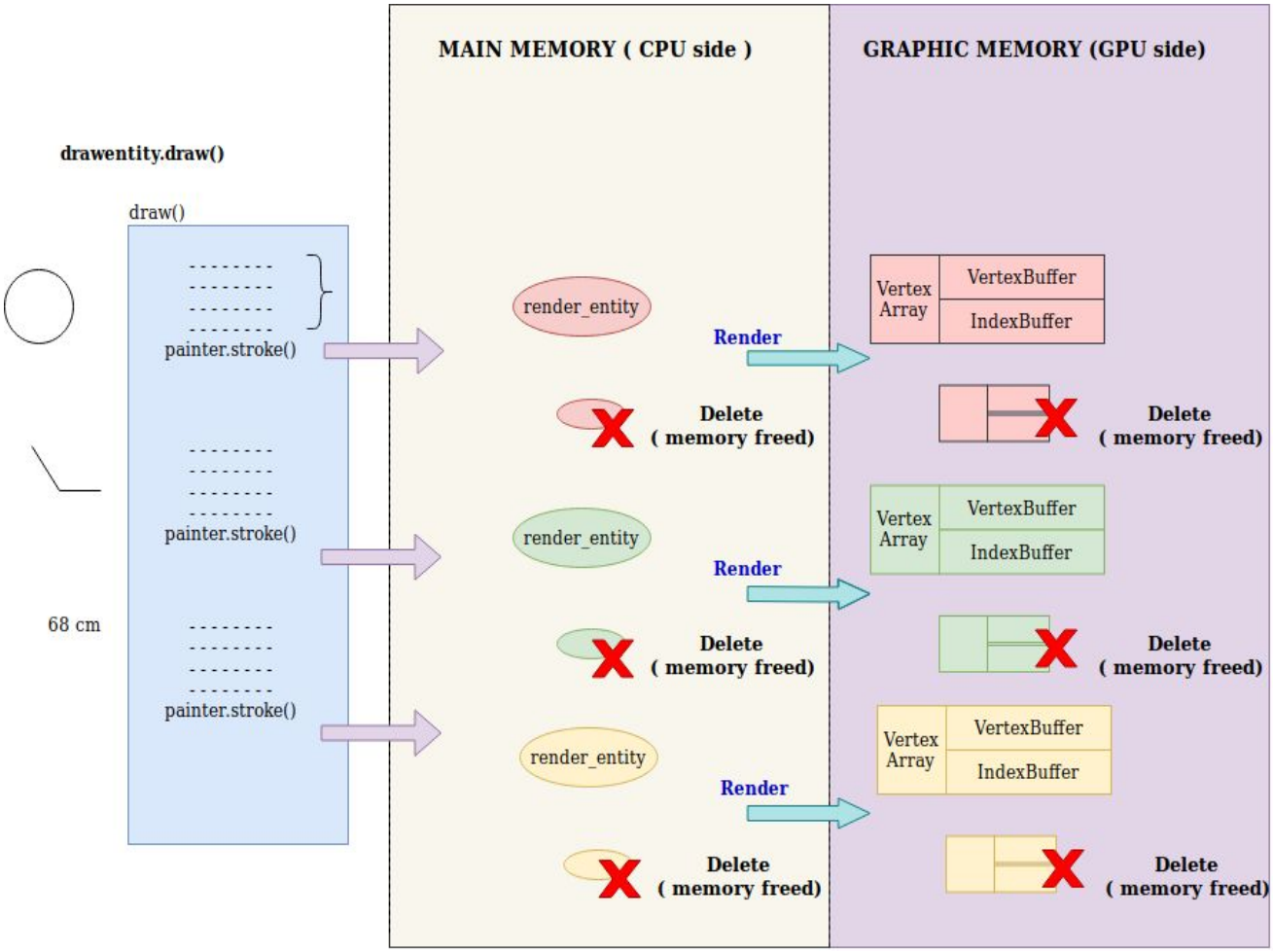
One of the major challenge with project is to let the librecad use its painter methods and still use the opengl rendering engine. At each **stroke()** the librecad painter draws (tends to) something.So correspondingly at each stroke() we have to send data from CPU to GPU (when entity is not cached)

The drawentities can be a group of simpler shapes, for each shape the data will be stored.The render_mode (GL_LINES ,GL_LINE_LOOP, GL_TRIANGLES) etc is set according to the need.



All those things which are not cached are drawn this way. The drawables (like cursor, grid etc) and the temporary entities which are in process of modification and needs frequent changes are rendered in this way. As they are modified so much at each frame saving them and editing them frequently can decrease performance.

As each time new BufferObjects are created and with interaction(s) after some time (after alot of refresh screen) the number of BufferObjects can increase so much which can lead to **memory leak** . So to avoid it the render will automatically delete the objects and free the memory from both **CPU and GPU.**

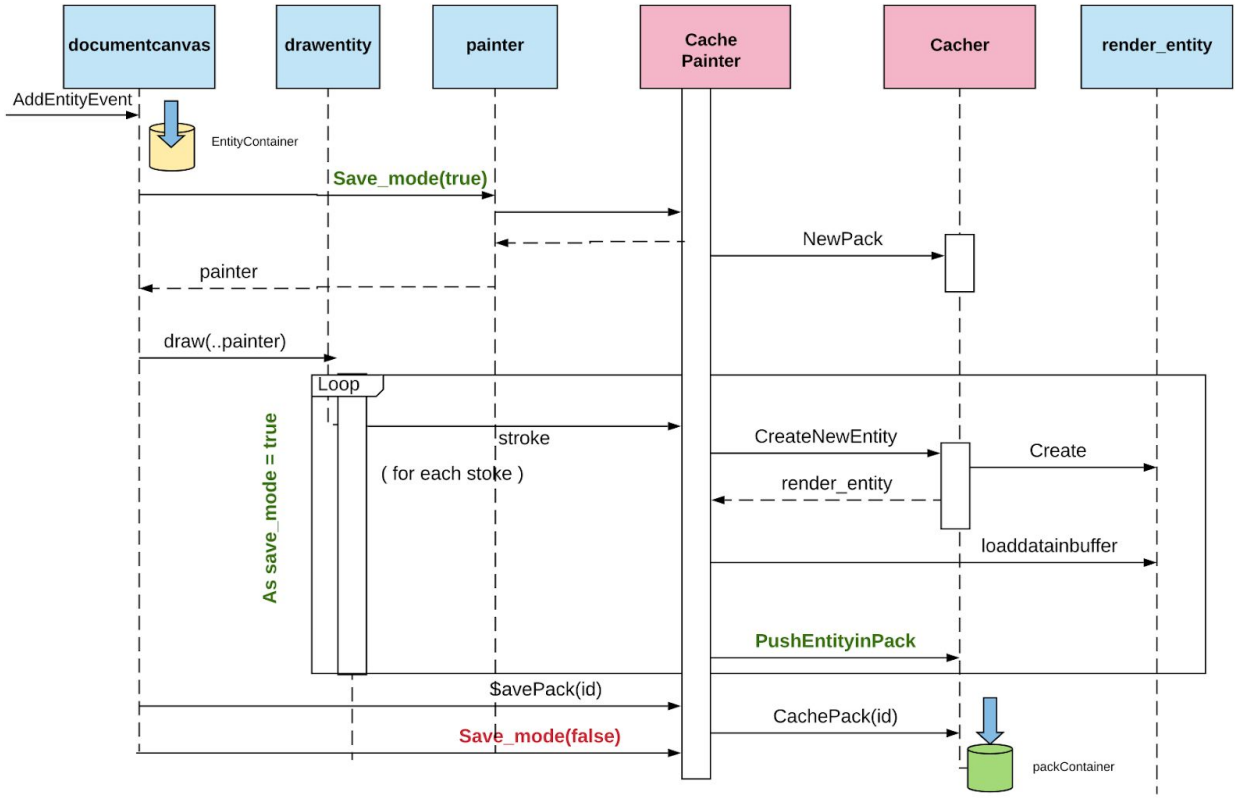


4. Caching Mechanism

In LibreCAD a number of shapes and entities can be drawn in a document. Most of the time these entities are not modified (or only few are modified at one time). So calling their draw() codes again and again in each frame is very inefficient.
So here comes the need of caching the corresponding buffer objects already set for entities at their creation and using them to render next frame. This will bypass the draw() and increase the performance.

When the draw() codes of drawentities are called to render nothing is saved. The opengl painter has another cachepainter inside itself when when used to send as parameter to onAddEntity() function of document canvas, the draw() code render nothing . Instead it prepares all the bufferobjects and saves them with the corresponding entity id.The cache painter is used so that the onAddEntity() be made **thread safe.**

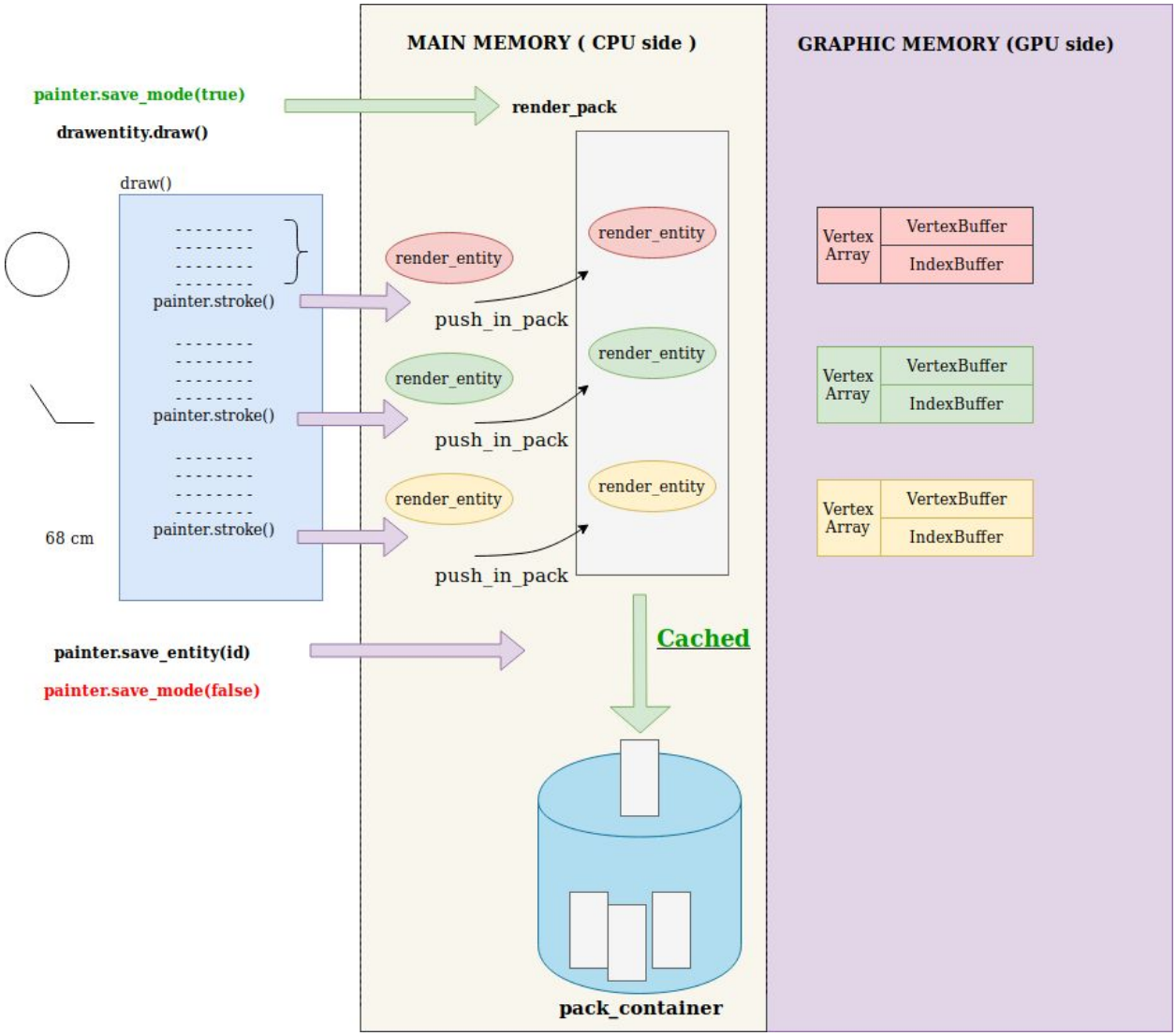
Below is the sequence diagram to show how entities are cached.



In these calls nothing is rendered only the data is send from CPU to GPU and the render_packs are cached.

Caching Mechanism: (Interactive Diagram)

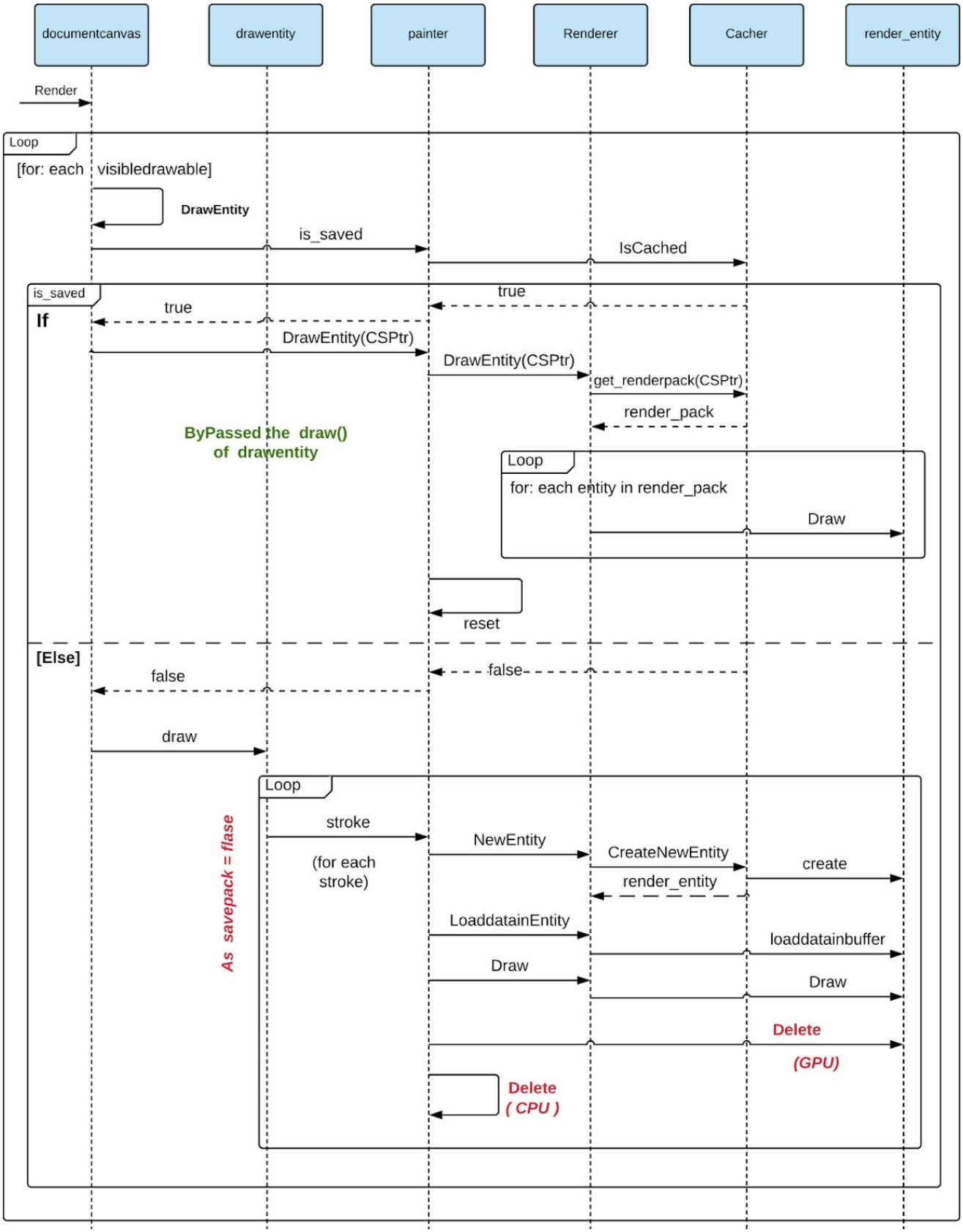
There is a structure render_pack which holds the group of render_entities. With each stroke() the painter **(cache painter)** push the prepared render_entity into render_pack and get ready for new entity.



Then in render loop the rendering becomes optimized, If an entity is already cached then its **draw() code** is completely **bypassed** and the already saved BufferObjects are Binded (accessed via render_pack cached) and rendered.

If an entity is not cached it follows the same routine again i.e, to call the draw() code prepare the buffers, render and then delete.

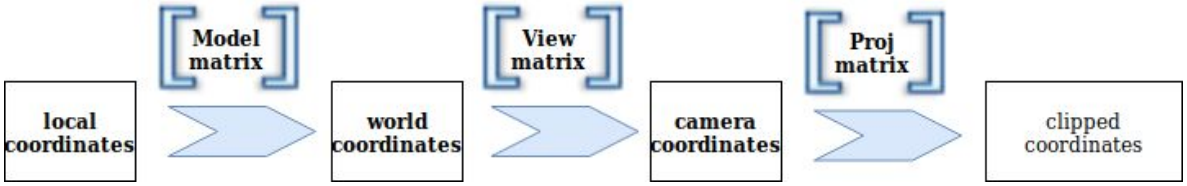
Here is the sequence diagram(for render loop):



5. Pan, Zoom and other transform facilities

As discussed earlier that OpenGL immediate mode and fixed pipeline should not be used. The depreciated direct calls like glTranslate(),glRotate(),glScale() and cannot be used with BufferObjects for zoom,pan or any other transform manipulations.

Rather model , view , projection matrices (or combine **MVP matrix**) should be applied on the **vertex data** by using uniforms in **shaders.** For matrix manipulations **glm library** is used. **ZOOM** is handled by the scaling matrix part of **model matrix . PAN** is handled by the translating the **view matrix. Projection matrix** handles the viewport , whenever resize event occurs it setups the project matrix with orthogonal matrix.



6. Text Rendering

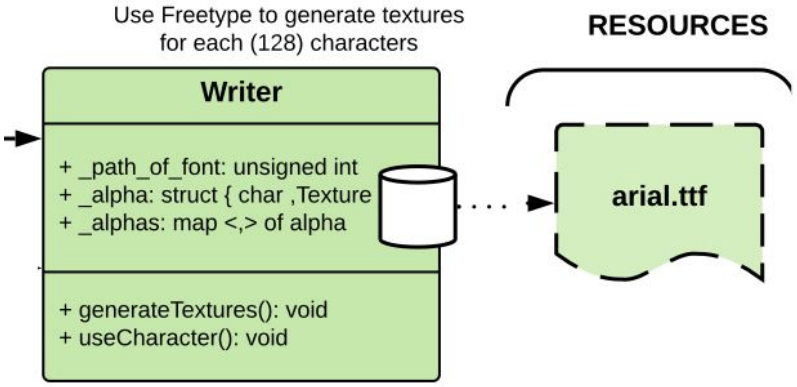
Since there is no support for any text capabilities within OpenGL it is up to us to define a system for rendering text to the screen.

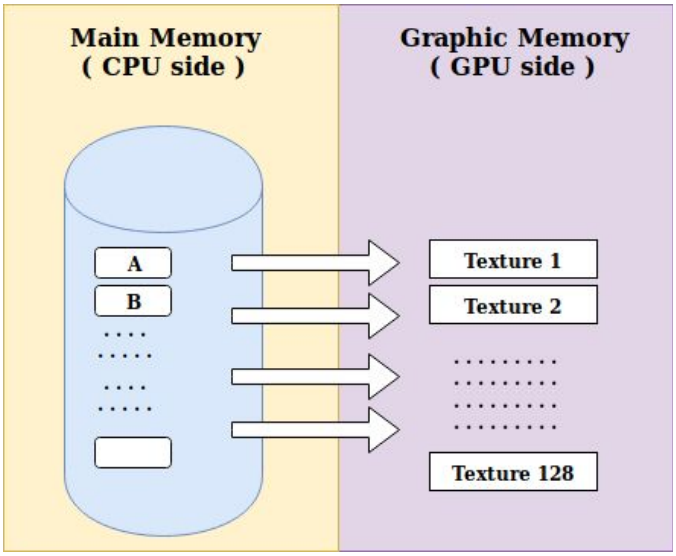
We have to use a texture **(bitmap font)** which contains all character symbols we want to use in predefined regions of the texture.The character symbols of the font are known as **glyphs**. Each glyph has a specific region of texture coordinates associated with them. Whenever we want to render a character, we select the corresponding glyph by rendering this section of the bitmap font to a 2D quad.

Why Freetype?

FreeType is a software development library that is able to load fonts, render them to bitmaps and provide support for several font-related operations.What FreeType does is load the TrueType fonts **(.ttf)** and for each **glyph** generates a bitmap image and calculates several metrics. We could load a character glyph, retrieve its metrics and generate a texture each time we want to render a character to the screen, but it would be inefficient to do this each frame.

We'd rather **store (128 characters)** the generated data in the **Writer class** and **query** (binding and applying to vertex data) it whenever we want to render a character this will **increase performance.**





7. Other features like gradient, line patterns etc

LibreCAD uses a **gradient background** for the document (there can be other places where gradient is needed).For this we require that the vertex data should bout only hold the coordinates but also the vertex color , the vertex buffer layout can be updated with this.And the shader (color vertex shader) can be used with this vertex data.With enabling the blending opengl will render the gradient surfaces.

To make stippled (dotted or dashed) lines, you use the command glLineStipple() to define the stipple pattern.The pattern argument is a 16-bit series of 0s and 1s, and it's repeated as necessary to stipple a given line.

Tentative Timeline

May 6, 2019 - May 27, 2019 (Community Bonding Period)

- I will continue with the **detailed implementation plan** of the opengl painter system .
- Making classes well abstracted and robust.
- Getting familiar of how the UI is connected with draw viewer area.
- Getting familiar with **upto what extent and edges** the lcpainter system can go and how it will be implemented.

May 27, 2019 - June 28, 2019 (First phase)

a) *May 27 - June 2:*

- Preparing well planned and abstracted classes for gl_entity, render_entity, VertexBuffer ,IndexBuffer ,VertexArray ,bufferlayouts.
- Making the Renderer class which is able to use these classes and draw shapes.
- Making Shader class, write some shaders in GLSL.
- Setting up correct context(s) for lcadviewer.

Milestone review:

➔ Correct working of the classes which are responsible to send data in GPU and creating Buffers. Also the correct opengl context.

b) *June 3 - June 9:*

- Setting up correct matrix manipulations using the glm library
- Painter features like device_to_user, translate,rotate,scale,pan and zoom

Milestone review:

➔ Correct working of the transformations using the matrices including the pan , zoom . Also the local model matrix for each entity.

c) *June 10 - June 16:*

- Implementing basic drawing shapes functions like line, rectangle, circle, arc , ellipse , fill ,close_path,new_path

Milestone review:

➔ Reviewing that the openglpainter class is compatible with the lcpainter system syntax

d) *June 17 - June 23:*

- Implementing the render_pack for those entities which are combination of more than on simple entity(like in lcv_dimension etc)

Milestone review:

➔ Reviewing that the openglpainter class can draw the combination type entities.

e) *June 24 - June 28:* **Phase 1 evaluation , getting feedback and making changes.**

June 29 , 2019 - July 26, 2019 (Second phase)

a) *June 29 - July 7:*

- Connecting the draw function of drawables and rendering them (cursor , grid ,background)
- Till Now the rendering is done without caching

Milestone review:

➔ Correct working of all the drawables.(except text rendering)

b) *July 8 - July 14:*

- Creating and attaching the caching painter with normal painter and the Cacher.
- Implement the caching mechanism.

Milestone review:

➔ Correct working of the caching mechanism.

c) *July 15 - July 21:*

- Connecting the rendering calls with the UI buttons

Milestone review:

➔ Reviewing that the rendering is done after choosing shapes from UI

d) *July 22 - July 26:*

- **Phase Second evaluation , getting feedback and making changes.**

July 27 , 2019 - August 26, 2019 (Third phase)

a) *July 27 - August 4:*

- Making Textures working in librecad.
- Working on the Text Rendering, implementing the Writer class and text_entity

Milestone review:

➔ Correct Texture and text Rendering in librecad

b) *August 5- August 11:*

- Working on gradient, hatches, gradient back ground.
- Working on bezier curves and implementing lcvspline

Milestone review:

➔ Correct working of the gradients and splines.

c) *August 12 - August 18:*

- Working on the line_patterns and Implementing remaining features.

Milestone review:

➔ Correct working of the line patterns

d) August 19 - August 26: (FINAL WEEK)

- Fixing the issues , cleaning the code and Optimisation (where possible)
- Getting review and submitting final work for evaluation.

(POST GSOC)

- Making code more better , modular and Optimized
- Keep maintaining it compatible with librecad (if required due to new features added in core)
- Keep maintaining the code and adding new features

Time availability

I can give 40 hrs / week easily and even more from the time duration of 15 May to July 31(Summer Holidays).
Once my Institute's new semester begin I can still give around 6-7 hours a day and 10-12 hours on weekends.

Why Me?

I have already been working on this project and has made a partial implementation of opengl rendering engine in LibreCAD 3,
Repository : **LibreCAD3_OpenGL_rendering** (which is not merged).

The already implemented features are:

- ✔️ Buffer Objects Abstraction
- ✔️ Renderer and Painter class
- ✔️ Renders basic shapes using buffer objects
- ✔️ Follows the lcpainter call system
- ✔️ Pan / Zoom

Can easily continue this project and make it complete.
Have a good knowledge of codebase of librecad and in good communication with mentors.

About Me

I am a undergraduate Computer Science student (Third year) pursuing B.Tech from **National Institute of Technology ,Hamirpur (NITH).**

I like doing Competitive Programming ,data structures , algorithm and problem solving. Very much passionate about graphic programming and developing computer games and softwares from scratch or game engines.I have also worked in the field of Virtual Reality and Augmented Reality (AR/VR).