GameApi Builder tool

Tero Pulkkinen
terop@kotiposti.net
meshpage.org
Tampere, Finland

ACM Reference Format:
Tero Pulkkinen. 2026. GameApi Builder tool. In *Proceedings of Web3D 2026 competition tools category (Web3D '26).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Abstract

GameApi Builder tool is a culmination of 14 years of software source code writing work by single person. It allows creation of 3d graphics in the web, designing scenes built from industry standard gltf and glb technologies to move, animate, change, edit properties, fine-tune and deploy beautiful 3d graphics using OpenGL/SDL2 and emscripten webassembly technology.

2 Type of tool

import sketchfab zip
import gltf
import glb
import obj
import stl
export html5 zip
export script.txt files
export obj files
editor of node graphs
viewer of 3d models
deploy of node graph
send to other computers via urls

3 Domain categories

3D Content Creation / Modelling
X3D Ecosystem / Web3D Tools
3D Content Publishing
Web3D User Experience

Thanks goes to Daniel Andersson(Bd-Calvin) for many useful discussions in irc.

Web3D '26,

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

4 Broader domain categories

game development
trailer creation
node graph tool
gltf standard implementation
emscripten / webassembly
3d web publishing
communication system
messaging over internet via urls
educational tools
target demographic: teenagers and high school students

5 What problem does this tool solve?

Teenagers and high school students need solid foundation to continue their exploration of computer games and computer graphics via proper 3d engine solutions. GameApi Builder tool is perfect stepping stone toward learning how 3d computer graphics is designed, created, deployed on the web and shared with their friends and family. The tool provides extensive node graph system that contains everything needed to make computer games, trailers, demos, 3d web publishing, 3d web pages and interactive content on OpenGL/WebGL canvas.

6 Software price and license

software price varies slightly dependending on where you get it from, but usually around $50-$150 dollars. License (after purchase) is LGPL and GPL.

7 Video

https://meshpage.org/web3d_video.mp4
https://www.youtube.com/watch?v=Hs7yMkjw-b0

8 Installation

https://gameapibuilder.com
https://meshpage.org
https://terop.itch.io/gameapi-builder
store.steampowered.com/app/4181720/GameApi_Builder/
https://github.com/terop2/GameApi

9 Software depenendencies

c++, php, js, emscripten, SDL2, tinygltf, stb_image, draco, glew, libfreetype, opencv videofile, httplib, normal browser environment, windows, linux

10 Platform support

10.1 Web platform

Web platform is the main platform where builder output is available. Pretty much all features should be working in web platform.

Emscripten compiler is needed to create web platform wasm files.

10.2 Linux platform

Builder tool itself runs in linux platform. All features work in linux platform.

Installation is slightly tricky in linux platform. The deb binaries only work in newest ubuntu release by default. This happens to be version 26.04 at the moment. If older ubuntu is available or non-ubuntu distribution, the alternatives are bleak: either get gpu working in docker image or use wine installation. Ubuntu binary compability is significant hurdle in releasing tools under linux platform. Upgrading to newer ubuntu is always an alternative, but do we want to force people to do that? Steam release has another way to handle the binary compability issue, it contains all the libraries coming from 26.04 release, and might have better compability.

gnu compiler is used in linux platform.

10.3 Windows platform

Builder tool itself also runs in windows platform. All features work in windows platform.

mingw compiler is used in windows platform.

10.4 Android platform

Android platform support is more limited. Many of the 3d scenes, especially biggger ones, fail in the android platform for memory related issues. Also many input methods including mouse and keyboard accesses are not available in android platform. Touchscreen support from builder tool is also very limited.

Android deliverables also require building the software from source code. Android ndk installation is required, and finding source code of sdl2 and libfreetype from the internet, and then compiling to android platform using provided scripts.

11 Software architecture

The software uses dll-free plugin architecture that uses dependency injection design using ctor parameters and reference data members. Constructors should describe all dependencies to a module, except some rare global variable uses and use of the base system like opengl and emscripten.

11.1 Types

Software functionality is based on c++ classes. From a class an instance is created using ordinary c++ `new`. That is passed to `add_()` function which moves the ownership of the object to `EnvImpl` module. This add function requires that the class is derived from polymorphic interface. `Envimpl` only stores pointers to base interface. The pointer is push_backed to a std::vector, and resulting index to the vector is an integer. That integer is wrapped to a struct: `struct P { int id; };` and that becomes `GameApi::P` handle. The software uses these handles to pass around objects. In next scope, `find()` function is used to convert the `GameApi::P` kind of handle back to the polymorphic interface pointer. `find()` function does not give ownership of the object to the caller, but instead keeps ownership inside `EnvImpl`. For this reason, `EnvImpl` needs to be created in software `main()` function, so that it has long enough lifetime.

11.2 Node graphs

From these types, conversions like `()->P`, `P->ML`, `ML->RUN`, `RUN->HML`, `HML->ZIP` are being done to build larger software from smaller pieces.

The path first loads assets and the resulting polygon mesh(`P`) is passed to the next node in the node graph. `P->ML` node is responsible of rendering the 3d model, resulting in a type and node that can be added to the event loop for listening software events. The event loop in `run_window::ML->RUN` will display a window and run event loop in it. `RUN->HML` will change platform from pure linux/windows system to a web based implementation. `HML->ZIP` will package the end result as a package file which can be uncompressed to web server directory and url to `display.html` file will be able to display the 3d scene described in the node graph.

11.3 Feature combinations

Important property of node graphs is that there are millions of combinations that are possible to be realized by connecting nodes together in different ways. Calculating the number of combinations from feature list perspective puts the number of possible combinations to 2^{860}, which is more than atoms in the observable universe. But type checking, i.e. type labels need to match/`P` type can only be connected to `P` type or `[P]` type, radically reduces the alternatives that are possible, but node parametrisation again grows the amount of possible outputs to extreme lengths. This is why interface consistency, i.e. how the types like `P`,`ML`,`RUN`,`HML` and `ZIP` are defined is extremly important for the library.

The connection itself is not doing anything except transfer of data between nodes. The nodes itself are doing all the hard work, and connections between nodes are just dummy data pipes between different plugin modules/nodes.

11.4 Envimpl

Envimpl just contains large number of std::vectors that keeps ownership of the objects handled in the library. Scope escape mechanism is however significantly more important feature of the EnvImpl than the ownership. So we strongly recommend to not refer EnvImpl as being garbage collection mechanism, but instead it should be considered scope escape mechanism that allows subgraphs to stay alive after the scope that created it was destroyed.

11.5 Base features

The base system in GameApi library contains mostly usage of external libraries like opengl and stb_image, and only small amount of actual behaviours. Most of the functionality is in the node graph. For this rule, there is only small number of notable exceptions, url loading and vector and matrix library is the important exceptions.

11.5.1 Url loading. Software can load urls from network using several different mechanisms. Linux and windows platforms use popen+curl approach to load urls from network. These are run in separate threads to speed up the loading. Web platform uses browser's own concurrent network stack. Url loading is able to give callback to the node graph to indicate that loading of a file is finished, and then node graph can continue doing operations based on the available data. This way the url loading is completely asyncronous.

11.5.2 Vector and matrix primitives. There exists quite comprehensive vector and matrix library that contain features like translate,rotate,scale of points and vectors using matrices or other linear transformations.

Notable features is the matrix inverse calculation, which is needed mostly for gltf animations code for transferring vertices from one coordinate system to another and back.

11.6 Prepare/execute separation

To get frame rates to work, operations involving vertex array data has been split to two parts: prepare phase and execute phase. Prepare is only called once in precalculation time, and it can do slow calculations. Execute phase should render it to screen, and that is very time critical operation, and programmers should move slow operations from execute to the prepare if possible.

11.7 Collect/heavyprepare

Prepare had a problem that it has all node graph nodes locked into single huge and slow operation using a node graph tree parent-child dependency. In browser environment this kind of slow operations that cannot be split to parts are completely killing drawing to the screen. `Collect` and `HeavyPrepare` -pattern fixes the rendering, especially for progress bar during prepare phase of the system initialization. This required moving the parent-child dependency to Collect function and keeping the slow operations in the different function, so that event loop can decide when to render progressbar screen between the slow operations.

11.8 Resize

Once 3d model has been loaded, we simply don't have information what coordinate system the 3d model was designed for. Resize feature tries to calculate bounding box for the 3d model and use its dimensions and position to move the 3d model to the screen in suitable size that it is clearly visible in the screen. Our screen uses coordinate system from $x, y, z \in [-450..450]$ range and 3d models are resized to be clearly visible in that area. Hunting 3d models location is pretty impossible if its not visible in the screen or rendered just 2 pixels in size like the 3d model's default coordinate system would require.

Note that for gltf animations, inverse of the resize is required or the rendering cannot move to correct coordinate system while applying transformations. This requires certain global pipeline in the base system where resize operation and gltf animation shader code can communicate via the pipeline. The exchanged data is simply matrices, but getting animation to the screen correctly without that data pipeline is next to impossible. The actual transformations happen in shader-side of the system, but software need to make the correct matrices available to the shader.

We have seen situations where resize is not using matrices, but more complicated non-linear logic is used to do resize operation. If inverse is not available, gltf animation simply cannot work. Best solution we've found is to disable the parts that are non-invertable, and try to use the invertable parts of the resize to render gltf animations. This kind of problems are nowhere to be found in our software solution (since we use linear matrix to do resize), and might only become problem once resize logic is further developed to non-linear approaches.

11.9 Event loop

The main event loop is divided to multiple parts. First it starts loading assets from the network, and event loop shows connecting and downloading logos to the user. Once all downloads finish, the software moves to prepare-phase, where collect/heavyprepare will prepare the system for rendering. Then it starts executing the rendering loop.

11.10 Deploy

To get output from the software, deploy feature is utmost importance. Node graph that developers generated with builder tool will be converted to format where html5 and web browsers is able to execute it, and deploy feature will output a html5 zip file that can be uncompressed to web server directory for browser access anywhere on the planet.

11.11 Http server

To get easy workflow and developers are able to see the result of their node graph in browser environment, `html_run` feature allows node graph to be directly displayed in the browser window. For this reason, a temporary http server is launched, and it uses local ip addresses to communicate with the browser. Node graph and related assets are transferred to the http server, and browser uses url to local network to do the transfer. Url_open feature is used to transfer url to the browser. Linux platform uses dbus to do this, windows have open-feature in operating system to do the url transfer.

12 Summary of existing features

12.1 Bitmaps

Bitmap features allow loading png/jpeg files and manipulate the resulting bitmaps in many different ways, including taking subbitmaps, scaling the bitmaps, changing colours, flipping the bitmap in x or y directions, filtering, rendering lightmaps, noise, and different kind of gradients, floodfill or creating world from multiple bitmaps.

In addition to rgba bitmaps, there exists boolean and float bitmaps with properties like modifying colours.

12.2 Polygon meshes

Meshes are very flexible objects in gameapi. The operations supported include gltf/glb, obj, stl model loading, extracting meshes from tinygltf data structures, rendering quads, heightfields, background images, polygons, cubes, rounded cubes, spheres, heightmaps, discs, cones, toruses, barcharts. Also more complex operations combining two meshes, translating, rotating, scaling mesh vertex arrays is supported (although considered slower operations than expected since it modifies all vertices of the mesh). Converting lines to cones is also supported.

Instancing support is also available, as in static instancing that copies the mesh to multiple copies. This is the slower cpu-side cloning of the instancing operation, which does not meet the performance requirements normally associated with opengl instancing.

Colour changes are supported. Normals can be calculated and smoothed. Flipping normals is supported. Choosing textures is also available. Sponza rendering needs texture splitter.

12.2.1 Facecollection and P type. The mesh data structure itself is an interface. It does not keep memory for full vertex arrays. Instead, mesh is built by fetching data from some existing mesh and applying some operations to it. This avoids significant memory usage and allows gameapi to use significantly larger 3d models than would otherwise be possible.

12.2.2 Or_elem and or_array. meshes can be combined. This feature requires that mixing of triangles, quads and polygons are available. This means that when we combine a mesh with triangles to a mesh that has quads, the result is a mixing of triangles and quads, and the low level rendering need to resolve the mix to separate arrays containing only triangles, or only quads or only polygons.

12.3 Gltf and glb

Figure 1 shows example render of glb files obtained from sketchfab.

The system provides gltf and glb support via the following paths:

```
gltf_load2 -> ml_gltf_all -> run_window
sketchfab_zip -> ml_gltf_all -> run_window
gltf_load2 -> ml_gltf_all_anim -> run_window
sketchfab_zip -> ml_gltf_all_anim -> run_window
```

Figure 1: Gltf rendering in web page

Gltf implementation supports gltf rendering, gltf materials, gltf animations. The following features are explicitly omitted: gltf cameras. Sketchfab compability should be in good shape, even though many gltf standard features are missing from the implementation. Animation support is still very limited, but phoenix_bird model and many similar models are working ok.

12.3.1 gltfmodelinterface and TF type. We have c++ interfaces to wrap tinygltf data structures. In builder tool the TF type represents tinygltf's data structure interface. Modifying the data in TF type is not really available at the moment, instead TF type is mostly for rendering glb files.

12.3.2 tinygltf usage. External tinygltf library is used to parse glb and gltf files, this is needed so that the end result is compatible with files available on the internet. There's no good reason to develop this functionality ourselves, given that library is available and developing it again just introduces bugs and incompabilities with the existing internet files.

However, tinygltf does not provide the rendering aspect, since it has dependencies to each individual 3d engine. Our 3d engine features are needed to do the rendering, and tinygltf library only provides data structures where the needed information is available for rendering. Actual opengl primitives are coming from our 3d engine.

12.3.3 gltf meshes. gltf rendering works with GltfFaceCollection class in the system.

12.3.4 gltf materials. gltf materials support is using our material system. Most of the sketchfab site gltf models work with the material system. Only small number of examples were found which fail in this area.

12.3.5 gltf animations. gltf animation support is another material in the material system. It needs a shader for animating the matrices. GltfShaderML class handles the animations. Gltf animation system has dependencies to the key events,

i.e. key events are used to activate animation sequences in the gltf animation system.

Gltf animation also requires pipeline for resize matrices to be passed from resize module to the gltf animation. These matrices are needed to transform back-and-forth between gltf file coordinate system and the coordinate system used for rendering. Inverse of resize matrix must exist.

12.4 obj and stl files

For sponza and sanmiguel rendering, obj loader is available. Converting obj files to .ds files is recommended for load-time performance reasons, as binary ds files have significantly less processing needed in load-time than parsing ascii .obj files.

STL files are mostly used for cad use cases. For trailers and mostly for gamedev purposes, stl files are pretty much outdated format. glb usage is preferred.

12.5 voxels and VX type

Voxels are also supported. One-colour voxels are done with VX type. There exists also multi-coloured version of voxels. MagicaVoxel .vox files are supported.

12.6 volume objects and O type

Volumeobjects are $(x, y, z) \rightarrow (bool, color)$ functions. These can be rendered many ways. Conversions to voxels are available.

12.7 font rendering

We also support font rendering primitives. Text can be rendered directly from libfreetype render functions, or from sprite atlas, or from our own glyph outline rendering algorithm that uses quadratic beziers and equation solving to render glyphs for text rendering. Only finnish alphabets are supported. English text might have problems with keyboard bindings, as finnish keyboard is currently the only thing supported.

12.8 boolean operations

There exists limited boolean operations implementation for 3d meshes. It has significant issues and applicability is very restricted.

12.9 mainloop api

Mainloop items are very flexible in gameapi-builder's node graph nodes. Everything that event loop can access needs to be mainloopitem which contains features for preparing the operations, and then executing frames and handling of the key events.

12.10 animation implementations

Our solution has multiple different animation implementations. First, gltf animation provides a way to do matrix animation based on splitting 3d model to smaller pieces using hierarchy in the glb files. Matrix stack is working via matrices in the shaders. Vertex animation which does interpolation of the actual 3d model points is another animation system. Sprite animations in 2d use feature for choosing different sprite frames for each displayed frame. Jumping from frame to another is also useful in 3d side, but should use larger time steps. Then move_ml feature can do matrix animations directly in the node graph.

Our best animation feature is combining the results of different animation systems. Gltf animation can be used together with jumping from frame to another. Vertex animations are more limited in usage, but could be useful to do slightly different kind of changes to the models.

12.10.1 gltf animation. gltf animation feature reads the skeleton from glb file and applies transformations using joints and weights. GltfAnimShader is used to run the matrices to the joints.

12.10.2 matrix stack animation. matrix stack just passes matrices to the shaders.

12.10.3 vertex animation – meshanim. simple `mix()` function in the shader allows linear interpolation of the vertices. However it has significant problems since all points change direction in the same frame, thus making it slightly bad for complex animations. But it could be useful for rendering tunnels or large corridors where direction changes are more controlled.

12.10.4 mlchooser. jumping from one frame to another is done via mlchooser. It allows choosing which ML type subgraph is used to render the scene. The t_ functions also provide similar functionality. This is important animation system derived from sprite animations.

12.10.5 move_ml. moving and positioning 3d models can be done via node graphs too, and `move_ml` provides that functionality.

12.11 opengl features

12.11.1 rendering. The library contains features where a 3d model from FaceCollection interface can be converted to opengl vertex arrays. FacePoint(), PointNormal(), Color(), TexCoord(), TexCoord3(), Joints(), Weights() gets fetched from the node graph with a copy-algorithm, and collected to vertex array std::vectors.

12.11.2 triangle-quad-polygon. Each vector has only triangles, only quads or only polygons. Opengl has separate mode for rendering `GL_TRIANGLES`, `GL_QUADS` and `GL_TRIANGLESTRIPS`, which allow rendering of mesh. An interleave algorithm is able to take mixed-tri-quad-polygon face soup and convert it to separate arrays for triangles, quads and trianglestrips. Threads are used to make the fetch from nodegraph -operation fast.

12.11.3 textures. To make 3d models look ok on the screen, texture mapping is used. For that reason, Bitmap¡Color¿ type BM is used to represent the pixels of a texture. Then it's fetched from the node graph, and passed to opengl. Threads are used to make the fetch operation fast.

12.11.4 instancing. Instancing allows copying 3d model several times to the screen without duplicating the vertex data. This requires either points data structure PTS, or matrices data structure MS to determine which location on the screen the new instances are to be placed.

12.11.5 level of detail (lod). Lod feature works similar way than instancing, but it divides the screen to lod levels, so that each level can have different accuracy for the drawn 3d model. This can speed up rendering 3d models located far in the world since all vertices do not need to be handled.

Our lod implementation depends heavily on availability of decimate algorithm. It uses percentage number to determine how much of the model is discarded/preserved, starting from smallest triangles or quads. The algorithm can leave holes to the model, if percentage is too small and only small number of faces are preserved. Each lod level uses different percentage number.

In low level side, critical feature for lod is the avaiability of dynamic count for the instancing instance count. For this reason, two numbers are needed, the dynamically changing number for each lod level. And static number, i.e. the maximum that will be used for instances in each lod level. Static number is used for memory allocation, while dynamic number is used for drawing. Using this information, the glBufferData() and glBufferSubData() calls can handle dynamic counts for the rendered instance count. The count is changing based on how many instances are in different areas of the view cone.

The view cone splitting to areas need to know the exact model view matrix handling in the library, so that area splitting operation works correctly. Because exactly correct solution is absolutely necessary, implementation of lod feature is often difficult. But it just needs to replicate already existing logic found in the model-view matrices.

12.11.6 shaders. Engine of course contains plenty of different shaders, organised as uber-shader which can be flagged with #ifdef's to contain different shader configurations. UberShader api handles the configuring the shader, but there exists separate shader mainloopitems that handle dynamic parameters to the shaders/pushes new parameter values every frame. Then materials usually collect the pieces needed together to form a full functionality suitable to be offered to the game developers.

12.11.7 shader combinations. Our shaders have small trick to allow combining shaders. The system automatically creates a chain of shaders where shader functions are called in a sequence and the needed positions for vertex shaders and rgba values from the fragment shaders are calculated in the chain. Each chain piece can use previous shader piece output and modify the result as is needed for that piece of functionality. This way shaders can be built incrementally with increasing complexity as more shader pieces are coming to life.

12.12 movement

MoveApi has significant number of primitives for translating, rotating and scaling of the 3d models using matrix stack. Animated versions anim_translate, anim_rotate, and scale are available to do dynamic movements to the world.

12.13 quake_ml

First-person movement operations are handling wasd and cursor keys movement in the world.

12.14 fly_ml

third-person movement needs access to the 3d model that player controls and then slightly more limited movement options are available compared to first-person movement.

12.15 opengl properties

opengl api provides many operations which are upgraded to developer control by providing high level operations that access opengl api directly. Disabling polygon rendering, depthfunction and depthmask, blendfunction, disabling shader cache, disabing zbuffer, transparent blending control, perspective and frame rate printing are provided.

Of utmost importance, is the transparent canvas operation. It allows in browser side disappearance of the video canvas. Transparent version of the canvas can be embedded to many web sites in such way that it seamlessly integrates to existing web page designs.

12.16 run_window

Run window is the event loop of the system.

12.17 html_run

Html_run converts existing node graph running in run_window to browser-side implementation.

12.18 save_deploy

Save_deploy extracts deployable html5 zip file from the node graph, in such way that uncompressing the zip file to web server directory and pointing browser's url to display.html file in the directory will display the 3d scene in the web browser.

Important use case is where iframe tag in html5 is used to point to the display.html file. This way 3d models can be embedded to customer's existing web pages without disturbing the overall structure of the html5 page. The iframe tag is usually in form `<iframe src="mydir/display.html" scrolling="no" width="830" height="630">`.

12.19 builder tool

Figure 2 shows how builder tool displays node graph.

Builder tool allows editing node graph, changing properties of each node, and then displaying the end result first in development window, and later in the full screen window, then in the browser window and finally as a html zip file deployed to the web server.

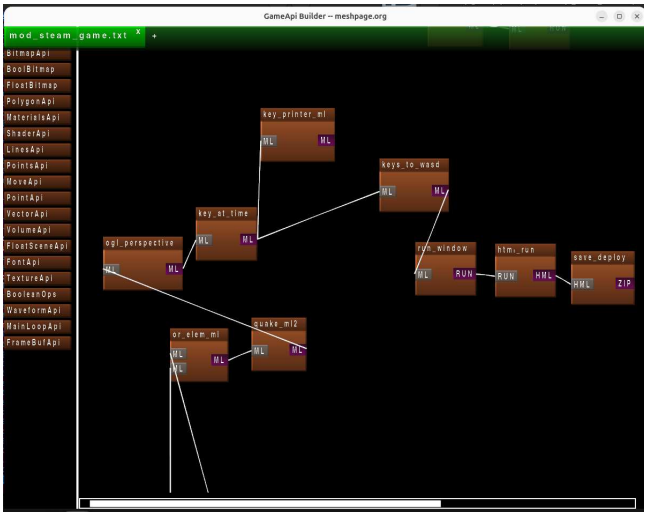

Figure 2: Builder tool user interface

12.19.1 node graph display. Largest area in the builder tool is reserved for node graph display. Node graph is displayed as boxes where you can connect the nodes using mouse when slot type labels match. After connecting, right-mouse button display menu provides ability to watch what the subgraph is able to produce 3d output.

12.19.2 caching and performance considerations. To keep builder performance working, caching is used at node load, so that repeated use of display menu is quick enough for efficient editing of the node graph.

12.19.3 list of features -display. At the left side of builder display, there is large list of features and possible nodes available to be selected and moved to the node graph display.

12.19.4 property dialogs. properties are opened from right-mouse button properties -menu. Each node in the node graph has all the parameters available that the software supports to be edited and fine-tuned to provide the best output possible.

12.19.5 property editors. each line in the property dialog allows editing single property.

12.19.6 drag and drop. Importing assets is done via drag and drop. For example, glb file, png file, script.txt file or any other file can be drag and dropped to appear in the download bar at the bottom of the builder display. Once there, the file can be further drag and dropped to the property dialog editor which contains urls to the files with the same extension like .glb or .png.

12.19.7 find. finding correct node can be difficult from set of 860 nodes. Thus find operation is available at the top of the feature list.

12.20 web integration

Web features are important to be of excellent condition. For this purpose, we use emscripten to allow execution of c++ code inside browsers. When deploy feature converts our node graphs to format suitable for web via html5 zip files. html_run feature is able to show the the node graph in browser window. The default display is using black video-like canvas display. For other than video-like uses of 3d graphics, we have transparent canvas which more readily blends to existing web pages. Then iframe tag can be used to embed the end result to article of your choice.

Figure 3: Sponza render

Figure 4: Sanmiguel render

12.21 sponza renders

Figure 3 shows how sponza model can be rendered.

Sponza rendering code handles the management of mtl and obj files and tries to find rendering properties that reproduce the old surface properties.

12.22 sanmiguel renders

Figure 4 shows how sanmiguel 3d model is rendered.

Important part of sanmiguel render is the ability to handle large amounts of vertices. The model doesn't use instancing

or any techniques like that, instead the instancing is implemented by copying vertices in the original source files, resulting in huge files and large amount of vertices/millions of vertices which slow down load time, rendering time etc. Ability to handle large amounts of data requires memory optimizations and performance optimizations.

12.23 videofile playing

Videofiles can be displayed. This works via texture id feature. There exists a texture id material that can render video frames. Actual video decoding is different on each platform: linux uses opencv library to decode video files, windows uses media framework from windows operating system, and web uses video tag available in the browsers. All these can output a video frame from the video file and we can pass it to texture id each frame and pass it to the 3d rendering.

12.24 feature registration

All nodes are created via three parts: c++ class that implements an interface, a function that creates an object, and registration line which registers the function to the system.

12.25 performance optimizations

Performance is important parameter for any 3d engine. Our solution is to use 6 different techniques to improve performance of the solution: performance profiling, caching, asyncronous loading, preloading of assets, concurrent download and threading.

12.25.1 browser's profiler use. The browser's profiler gives view of where bottlenecks are located. Our principle is to optimize only those areas which are visible in the performance profiling. Half year of finding issues with the performance setup with the principle allows for good performance for the 3d engine.

12.25.2 caching. Builder tool has important performance bottleneck near the loading of assets. The data coming from internet takes long time to load from the network, and to avoid this performance problem when display-menu in builder tool is invoked, caching provides the best solution. First time load is still slow, but caching improves the loading of the same assets in 2nd and later loads, assuming the asset set stays the same.

12.25.3 asyncronous loading. Multiple files are loaded from the network at the same time, and the data will arrive from the network in random order. For this reason, asyncronous loading is used, and the system gives callbacks to nodes in the node graph when url loads are finished. With the callback further loads are able to start when new files to be loaded are needed based on freshly loaded content. For example, script file loading might have additional urls embedded in the script that will need to be loaded after script loading is finished.

The system will wait and display progress bar and logo while assets are being loaded. Once all assets are loaded, the system lets script itself to display content to the screen.

12.25.4 preloading of assets. Asset loading is started as early as possible, to avoid load time delays.

12.25.5 concurrent download. Originally developed for the sanmiguel rendering, concurrent download feature is now enabled by default. It splits loading the small chunks and loads large number of chunks at the same time. This improves the load time since network packet sizes can stay small, giving priority to our packets in the internet.

12.25.6 threading. To enable use of multiple cpu cores, threading is used. Browser requires SharedArrayBuffer usage to enable pthreads in emscripten. SharedArrayBuffer only works once https is available. And https requires obtaining domain name from the internet service providers. The deployed software package has been divided to two parts, threaded version is loading different files compared to no_thread version.

Threads help with uncompressing sketchfab zip files, parsing gltf/glb files, decoding textures, uploading textures and vertex arrays to gpu, fetching the vertex data from node graph.

12.26 memory usage optimizations

Memory usage is also very important for 3d engine. The optimizations include using vertex arrays in stack and fetching them in slices, moving vertex arrays to inside c++ type system, and freeing memory immediately after passing data to gpu memory.

12.26.1 vertex arrays in stack. Vertex arrays are move through the node graph in small one-vertex sized slices in stack. This allows modifying the vertex data and data amount while it is being transferred through the nodes. This is very big memory optimization, since nodes in the node graph no longer need to allocate memory for full vertex array, but instead stack deallocates the data automatically and memory usage never grows very large due to simply passing around vertex arrays.

12.26.2 fetch slices. A vertex array slice is one-vertex sized slice from vertex array fetch, through the node graph from the root toward the loading of the data from the network. This is repeatedly called for all vertices until all data has been obtained.

12.26.3 c++ type system and sizeof problem. sizeof(T) in c++ has slightly bad feature for all 3d engines using opengl. Since opengl vertex array size is dynamic, and c++ types have compile-time size, there's no way to put full vertex array inside a c++ type. Hacks are normally used to avoid this problem, including placing vertex arrays to heap when they are outside of c++ type system. To avoid this, our system splits vertex arrays to slices, and ensures that every slice has compile-time size. Thus FaceCollection fetch functions FacePoint, PointNormal, Color, TexCoord, TexCoord3 etc will need to have return value that passes only very small piece of the vertex array at each function call. Usually just single vertex worth of data is passed in each function call.

The node graph has dependencies to other nodes. Thus the function call will do multiple forwarding calls until it can find the original source of the data and can obtain the needed data. This causes slight execution jumping across the node graph millions of times. However our performance measurements have shown that this isn't significant performance problem, as it become visible in profiling only very late in the profiling activity. No good optimizations are available for this, but it's still significantly better than placing vertex arrays to heap during node graph transport.

12.26.4 freeing memory immediately after passing data to gpu mem. Opengl requires the vertex arrays in large arrays. So slices will need to be converted to actual opengl vertex arrays. Memory optimization is to allocate the vertex array memory, pass the data to gpu memory, and then immediately free the heap memory required. This way, vertex array memory is only needed very short amount of time, and overall memory usage never grows past the limits.

12.27 build scripts

12.27.1 Makefiles. Normal makefiles have 3 different modes of operations. Fast mode builds the source code as quickly as possible, without recards of if the whole package is ready after compile. Depbuild mode does ordinary dependency build which should handle full dependency graph of the source code. Then cleanbuild does full build. And clean does remove extra files.

12.27.2 emmake.sh. emmake handles emscripten build

12.27.3 ammake.sh. ammake handles android build

12.27.4 full_deploy.sh. full deploy build the system in linux side and pushes the output to our web server.

12.27.5 full_deploy_win.bat. windows side has also full deploy script

12.27.6 steam_deploy.sh. steam deploy does linux side steam build.

12.27.7 copy_to_steam.bat. windows side copying of the linux side data to steam's data structures

12.27.8 copy_to_steam_win.bat. windows side building and copying of windows side data to steam's data structures

12.28 cmdline tool

Additionally, there exists gameapi_cmdline tool for running end result from builder in linux and windows platforms. cmdline tool is receiving little testing and could have significant issues that have not been fixed including at least problems with loading file:// urls coming from builder drag and drop.

13 Controlling gaming logic

Gaming logic features are very limited in the library. This is due to every game requiring different kind of gaming logic, and finding reusable pieces from that mess is very difficult. Our solution to the problem is providing the source code for the engine, so that game developers can compile the source code themselves, start creating their own c++ classes and object creation functions and then registering the resulting functions to the system. This requires dealing with 3d engine internals. This might require advanced knowledge of the c++ and related technologies. Full usage of this with deploy working requires usage of emscripten compiler and is considered advanced usage especially for our intended user demographic. Github repo has instructions how to do this.

14 Conclusion and further developments

Current status of the tool is that it seems pretty stable, even though we have only limited exposure to real users using the system. Definitely users will find mistakes and errors and user interface quirks from the software when they start using it, but we hope the remaining problems are not significant enough to prevent the usage of the system as a nice 3d web system with ability to deploy 3d models as html5 packages to a web server. We recommend especially starting game developers from teenager and high school student communities to try our tool.