omigroup / **gltf-extensions** Public

Code Issues 8 Pull requests 4 **Discussions** Actions ···

glTF Collision Extension #50

RangerMauve started this conversation in **Ideas**

RangerMauve on Jan 10, 2022

Hey folks, I wanted to follow up on last week's meeting where we started talking about creating an extension for collision meshes in GLTF scenes / objects.

Goals

Goals of the Discussion

- Look at prior art and what can realistically be ported between major game engines
- Discuss what the JSON structure could look like
- Get signals for who's down to write code for export/import and game engine loaders.

Goals of the Extension

- Specify a GLTF extension that will enable reusable collision detection for objects across metaverses
- Focus on a high level API and reuse existing concepts as much as possible
- Be able to degrade to a non-extended standard GLTF (avoid VRM issues)

Non-goals of the Extension

- Physics-specific properties
- Navmesh (separate proposal after?)

Prior art

Godot

Godot currently supports automatically detecting collision meshes in GLTF scenes by looking at suffixes in the name of a given object.

This works by detecting appropriate suffixes on meshes (e.g. `-col, -convcol, -colonly`) and attaching the appropriate colission node to the entity. The `-col` suffix represents a full Mesh colission node which is precise, but also expensive to run, the `-convcol` can be added to a mesh to hint that it's to be a ConvexPolygonShape which has performance improvements for convext polygons. The `-colonly` flag will remove the mesh itself and just leave the StaticBody colission object in its place.

When using `-colonly` on "empty objects", you will be able to specify a primitive shape for collision between "arrow", "box", "plane", and "sphere". (TODO: what are the parameters for each). These primitives are a lot more efficient for collision detection so developers should generally aim to use them instead of meshes whenever possible.

With the `-rigid` suffix, a node can be set as a `RigidBody` which can be configured to be `static` or `rigid/character/kinematic` (all of those basically mean non-static).

Mozilla Hubs

Hubs uses their own extension called ammo-shape which define a type of collision shape for the object.

They specify a set of types (`box` , `sphere` , `hull` , `mesh`) which can be attached to an object.

They also specify a way to automatically fit an object at runtime based on the objects vertecies.

It looks like this is applied to an entire object at once and there's no way to specify individual colission shapes within an object.

Potential paths

I don't think we should adopt the suffx-based standard from Godot, but I think this gives a good idea of the sorts of things we'll want to consider for this proposal. I'm also not sure if the Hubs approach will scale well for complex objects that might have different bits and pieces.

Godot-style

I think a good starting place would be the ability to specify that a node is supposed to be used for collision only by giving it the `omi-collision` extension which can specify whether it's a mesh (if it is, link to the mesh ID), or specify that it's a `primitive` in which case one can specify what type of primitive it is an the parameters for it (e.g. box, sphere).

I think these nodes should not be displayed graphically and should only be used for collision. If you also want a graphical display, then you should be able to add that to the parent node. One thing to figure out is how a renderer which doesn't support these nodes will handle them, or how Blender could convert nodes in it's scene graph to be collision shapes for the GLTF.

A node can have any number of these collision objects inside them. One question is whether we'll need to make any sort of distinction for the parent node along the lines of it being a RigidBody or not (or if it can be punted to a future physics discussion).

Hubs-style

Alternately, it might be easier for developers if we go the hubs-style where the engine can compute the collision for a visible item at runtime with a higher level extension that looks like the Hubs one.

This would mean an object would get wrapped with it's collision, and I'm not sure how it could be expressed with the scene graph in something like Blender (could a child sphere node get marked somehow to be converted into a sphere collision shape for its parent?)

It seems this was mostly meant to be used at the top level of a "scene" object for an avatar so that it can collide with the environment rather than having several objects in a scene be collision-enabled.

Potential uses

With collision meshes in place, it could be easier for us to experiment with what physics or interaction could look like.

I could imagine a case where you have a complex GLTF object which has buttons on it which can trigger "collision" events in a WASM script that will make it react.

It might also be a path to people starting to experiment with turning objects into RigidBodies (or the such) so that you can spawn a map with a basic collision mesh and keep players and objects from falling through it when you imbue the with physics.

TODO:

It'd be nice to get some feedback on which paths people think would be most useful to use as a group and if there are other paths we should consider.

Anotherr thing that would be handy is if we could get more examples of importing and exporting collision meshes in editors and game engines to steer our focus.

↑ 4

3 comments · 3 replies

Oldest Newest Top

AdamFrisby on Jan 12, 2022

Pardon the dumb question, but does GLTF allow leaf nodes storing arbitrary data? A collision leaf childed below nodes could store component information about the format of the collider, size, shape and offsets and so forth.

↑ 2

1 reply

RangerMauve on Jan 12, 2022 Author

Would the parent in this case be marked as having some sort of collision? Or is it just that the child nodes are all enabled with collision?

AFAIK all nodes in the scene can have arbitrary extensions added to them.

robertlong on Jan 13, 2022 Maintainer

edited

Notes from 1/13 Meeting:

- Auto generate colliders from a given mesh?
  - Configure to auto generate or specify trimesh directly
- Allow for trimeshes?
- Only convex mesh contraints?
- Trimeshes use `mesh` vs `accessor` data?
- Prior Art:
  - VRM https://github.com/vrm-c/vrm-specification/tree/master/specification/VRMC_springBone-1.0-beta#vrmc_springbonecolliders
  - Virtual Cast https://github.com/virtual-cast/VCI/blob/master/Assets/VCI/UniVCI/Scripts/Format/GltfExtensions/NodeExtensions/glTF_VCAST_vci_colliders/PhysicMaterialJsonObject.cs
  - Hubs
    - https://github.com/MozillaReality/hubs-blender-exporter/blob/bb28096159e1049b6b80da00b1ae1534a6ca0855/default-config.json#L608
    - https://github.com/mozilla/hubs/blob/master/src/gltf-component-mappings.js#L46
    - https://github.com/mozilla/hubs/blob/master/src/gltf-component-mappings.js#L439
    - https://github.com/mozilla/hubs/blob/master/src/gltf-component-mappings.js#L428
    - https://github.com/mozilla/hubs/blob/master/src/gltf-component-mappings.js#L56
  - https://github.com/kmammou/v-hacd
- Usecases
  - Props (used for picking up an object, bumping into it, colliding with environment / avatars)
  - Environment (Colliding with environment, trigger volumes)
- For trigger volumes some engines may need colliders to be primitives?
- Possibly table physics engine specific properties and make into separate extensions. Focus on geometries? Or do we need to keep some physics properties in mind for this spec?

- nonphysical, static, dynamic, kinematic ?

↑ 1 1 reply

RangerMauve on Jan 13, 2022 Author

More detailed notes about what we talked about here: #49 (comment)

avaer on Jan 13, 2022 edited

Summary

(opinion) The minimum useful outcome for devs and artists is to be able to specify, declaratively, collider(s) on an object, which is an 80-90% solution. The concrete specifics of that (e.g. how meshes are decomposed, how to fall back for performance) seem better left to implementers and `SHOULD` in the spec.

After V1 of the spec ships, it makes sense to explore things beyond just definitions of shapes.

Background

Almost all physics engines support the same high level set of colliders, the same concept of transforms, and the same basic parameters (like the bounds of a box, or the radius of a capsule), so a standard for "this object has these colliders" is easily implemented in any engine if you choose the correct set of objects.

I could be wildly off, but I think this is a pretty complete set of everything you are likely to see:

- Box (size)
- Plane (normal, distance)
- Capsule (radius, height)
- Trimesh (triangles)
- Convex/hull (triangles)

Defining these, and the answers to questions like "Which way is a capsule pill rotated?" and "Does a box start at the bottom or in the center" would be a great start for interop across engines!

There are tangentially related physics properties that are common, like the friction and restitution materials for colliders, but I'm not sure if that has the kind of strong agreement across all engines like the above set.

Collider types that are often supported and used but I'm not sure have enough de-facto agreement to be in a standard:

- Heightfield (2d array)
- Voxel field (3d array)

Extensions

Physics engines often have other tweakable properties on physics other than just shapes and positions. These seem like good ideas for future extensions.

- Velocity animation
- Skinning
- Materials (slippery?)
- Triggers (bonk head)
- Gravity
- Kinematics

- Attachment (e.g. wearable)

Standard buy-in

Unlike definitions of colliders, there are significant and legitimate engine/gameplay disagreements about user interactions, mechanics, optimization, and performance of physics. That makes it a difficult proposition to have any opinions about that in the standard.

Games+engines will always optimize for their own users and usage regardless of what the standard will say. However the baseline definition seems like it would be pure 👌 for everyone, since it is easily implemented and does not dictate the rest of how your game should function.

Next steps

Dream starting point for me would be a very simple GLTF metadata standard for a set of colliders similar to the above list, with the intention to prototype and see if there is anything else that is universally needed in such a standard.

Webaverse would gladly contribute mapping code for THREE.js GLTF -> PhysX WASM.

↑ 3 1 reply

RangerMauve on Jan 18, 2022 Author

Would you be interested in proposing a shape for what the extension metadata should look like? I'm also down to do this and bikeshed in a GitHub PR. (probably next week?)

Could discuss this more at the next meeting.

Category

💡 **Ideas**

Labels

None yet

4 participants