OMI_physics_shape

Contributors

- Aaron Franke, The Mirror Megaverse Inc.
- Robert Long, The Matrix.org Foundation
- Mauve Signweaver, Mauve Software Inc.

Status

Open Metaverse Interoperability Group Stage 1 Proposal

Dependencies

Written against the glTF 2.0 spec.

Does nothing on its own. Designed to be used together with the `OMI_physics_body` spec.

Overview

Physics shapes can be added to glTF nodes along with information about the "type" of shape it is representing.

This extension allows specifying physics shapes to be used in glTF scenes. `OMI_physics_shape` can be added to glTF nodes, which references the document-level list of physics shapes. They are meant to be used by `OMI_physics_body`. Without a body or another extension using it, this extension does not mandate any particular behavior for the nodes aside from their geometric shape. The precise usage of these physics shape primitives SHOULD be specified by the extenions which utilize them. In general, these physics shapes are used to specify geometry which can be used for collision detection.

The `OMI_physics_body` extension specifes the behavior of attached shapes, including static, kinematic, rigid, and non-solid triggers. Implementations MUST also implement `OMI_physics_body` to determine the behavior of the shapes, or else the shapes do not have defined behavior. Even for a scene with physics shapes that do not move, the body extension contains crucial information about how the shape should be treated, notably whether the shape is solid or not (a trigger body's shapes are not solid).

Example:

This example defines a single box shape with a size of 1 meter in all dimensions as a child of a static body:

```
{
    "asset": {
        "version": "2.0"
    },
    "extensions": {
        "OMI_physics_shape": {
            "shapes": [
                {
                    "size": [1, 1, 1],
                    "type": "box"
                }
            ]
        }
    },
    "extensionsUsed": [
```

```
        "OMI_physics_body",
        "OMI_physics_shape"
    ],
    "nodes": [
        {
            "children": [1],
            "extensions": {
                "OMI_physics_body": {
                    "type": "static"
                }
            },
            "name": "StaticBox"
        },
        {
            "extensions": {
                "OMI_physics_shape": {
                    "shape": 0
                }
            },
            "name": "BoxShape"
        }
    ],
    "scene": 0,
    "scenes": [
        {
            "nodes": [0]
        }
    ]
}
```

More example assets can be found in the examples/ folder.

glTF Schema Updates

This extension consists of three new data structures for defining physics shapes on the root glTF document and referencing them on a glTF node. The main data structure defines a physics shape and is what most of this document describes. The second data structure uses the key `"OMI_physics_shape"` in the document-level `"extensions"` which contains a list of the main physics shape data structures. The third data structure uses the key `"OMI_physics_shape"` in the node-level `"extensions"` which contains an index of the physics shape to use from the list document-level physics shape list.

The extension must also be added to the glTF's `extensionsUsed` array and because it is optional, it does not need to be added to the `extensionsRequired` array.

Property Summary

The rest of the document, including this summary, defines the properties for the main data structure.

	Type	Description	Default value	Valid on
type	`string`	The type of the physics shape as a string.	Required, no default	Always valid
size	`number[3]`	The size of the box shape in meters.	[1.0, 1.0, 1.0]	Box
radius	`number`	The radius of the shape in meters.	0.5	Sphere, capsule, cylinder
height	`number`	The height of the shape in meters.	2.0	Capsule, cylinder

	Type	Description	Default value	Valid on
mesh	`number`	The index of the glTF mesh in the document to use as a trimesh shape.	-1	Trimesh, hull

Shape Types

The `"type"` property is a lowercase string that defines what type of shape this physics shape is.

The selection of shapes was carefully chosen with a balance of compatibility between major game engines and containing the most commonly used shapes for easy asset creation. Physics shapes inherit the transform of the glTF node they are attached to. This includes rotation and translation, however it is discouraged to scale physics shape nodes since this can cause problems in some physics engines.

Here is a table listing the mapping between the `OMI_physics_shape` type and the equivalent types in major game engines.

Shape	Unity	Unreal	Godot	Blender	Bullet (Ammo, Panda3D, etc)
Box	Box	Box	BoxShape3D	Box	Box Shape
Sphere	Sphere	Sphere	SphereShape3D	Sphere	Sphere Shape
Capsule	Capsule	Capsule	CapsuleShape3D	Capsule	Capsule Shape
Cylinder	Approximation	Approximation	CylinderShape3D	Cylinder	Cylinder Shape
Hull	Mesh (Convex)	Convex	ConvexPolygonShape3D	Convex Hull	Convex Shape
Trimesh	Mesh	Mesh	ConcavePolygonShape3D	Mesh	Mesh Shape

Box

Box shapes describe a cube or cuboid shape. They have `size` property which is an array of 3 numbers that describes the width, height, and depth. If the `size` property is omitted, the default size is `[1, 1, 1]`, representing a cube with a volume of one cubic meter, edges/diameters one meter long, and extents/radius of half a meter. The position of the glTF node is the center of the box shape.

Sphere

Sphere shapes describe a uniform "ball" shape. They have a `radius` property which is a single number. If the `radius` property is omitted, the default radius is `0.5`, representing a sphere with a radius of half a meter, a diameter of one meter. The position of the glTF node is the center of the sphere shape.

Capsule

Capsule shapes describe a "pill" shape. They have a `radius` and `height` property. The height is aligned with the node's local vertical axis. If you wish to align it along a different axis, rotate the glTF node. If the `radius` property is omitted, the default radius is `0.5`, and if the `height` property is omitted, the default height is `2.0`. The height describes the total height from bottom to top. The height of the capsule must be at least twice as much as the radius. The "mid-height" between the centers of each spherical cap end can be found with `height - radius * 2.0`.

Cylinder

Cylinder shapes describe a "tall circle" shape. They are similar in structure to capsules, they have a `radius` and `height` property. The height is aligned with the node's local vertical axis. If you wish to align it along a different axis, rotate the glTF node. If the `radius` property is omitted, the default radius is `0.5`, and if the `height` property is omitted, the default height is `2.0`.

The use of cylinder is discouraged if another shape would work well in its place. Cylinders are harder to calculate than boxes, spheres, and capsules. Not all game engines support cylinder shapes. Engines that do not support cylinder shapes should use an approximation, such as a convex hull roughly shaped like a cylinder. Cylinders over twice as tall as they are wide can use another approximation: a convex hull combined with an embedded capsule (to allow for smooth rolling), by copying the cylinder's values into a new capsule shape.

Hull

Hull shapes represent a convex hull. Being "convex" means that the shape cannot have any holes or divots. Hulls are defined with a `mesh` property with an index of a mesh in the glTF `meshes` array. The glTF mesh in the array MUST be a `trimesh` to work, and should be made of only one glTF mesh primitive (one surface). Valid hulls must contain at least one triangle, which becomes three points on the convex hull. Hulls are recommended to have at least four points so that they have 3D volume. The final hull shape should have no more than 255 points in total.

Hulls can be used to represent complex convex shapes that are not easy to represent with other primitives. If a shape can be represented with a few primitives, prefer using those primitives instead of convex hulls. Hulls are much faster than trimesh shapes.

Trimesh

Trimesh shapes represent a concave triangle mesh. They are defined with a `mesh` property with an index of a mesh in the glTF `meshes` array. The glTF mesh in the array MUST be a `trimesh` to work, and should be made of only one glTF mesh primitive (one surface). Valid trimesh shapes must contain at least one triangle.

Avoid using a trimesh shape for most objects, they are the slowest shapes to calculate and have several limitations. Most physics engines do not support moving trimesh shapes or calculating collisions between multiple trimesh shapes. Trimesh shapes will not work reliably with trigger bodies or with pushing objects out due to not having an "interior" space, they only have a surface. Trimesh shapes are typically used for complex level geometry (for example, things that you can go inside of). If your shape can be represented with a combination of simpler primitives, or a convex hull, or multiple convex hulls, prefer that instead.

JSON Schema

See schema/shape.schema.json for the main shape schema, schema/glTF.OMI_physics_shape.schema.json for the document-level list of shapes, and schema/node.OMI_physics_shape.schema.json for the node-level shape.

Known Implementations

- None

Resources:

- Godot Shapes: https://docs.godotengine.org/en/latest/classes/class_shape3d.html
- Unity Colliders: https://docs.unity3d.com/Manual/CollidersOverview.html
- Unreal Engine Collision Shapes: https://docs.unrealengine.com/4.27/en-US/API/Runtime/PhysicsCore/FCollisionShape/
- Unreal Engine Mesh Collisions: https://docs.unrealengine.com/4.27/en-US/WorkingWithContent/Types/StaticMeshes/HowTo/SettingCollision/
- Blender Collisions: https://docs.blender.org/manual/en/latest/physics/rigid_body/properties/collisions.html
- Mozilla Hubs ammo-shape: https://github.com/MozillaReality/hubs-blender-exporter/blob/bb28096159e1049b6b80da00b1ae1534a6ca0855/default-config.json#L608