ONTOLOGY FOR MEDIA CREATION

PART 3D: CG ASSETS

VERSION 2.8

Motion Picture Laboratories

, 2025

Contents

1 Introduction

Computer Graphics (henceforth CG, also known as CGI – Computer Generated Imagery) has been used in film production for much longer than people realize. It can be argued that it first appeared in the title sequence for *Vertigo* (1958).[1] *Westworld* (1973) was the first that used it in combination with live-action. After that came *Star Wars: Episode IV* and *Tron*, after which it was used more and more frequently, up to *Toy Story* (1995), the first full computer-generated feature-length film. Now is it not uncommon for VFX heavy feature films to have hundreds of shots, and CG animated films represent over 85% of the animated content released in a given year.

CG is pretty much omnipresent in modern film and television, and introduces a great deal of complexity. OMC support for CG is not intended to cover all aspects of it. Rather, OMC CG Assets focus on three things:

- The connection of the CG workflow to the rest of the production process. OMC manages this through context and relationships, much of which applies just as well to non-CG work. For example, CG models can be used in depictions, as pre-production art, and as components of VFX sequences – all standard OMC relationships.
- Understanding the high level representation and dependencies of CG Assets. As CG Assets are tied to the larger production process and shared between workflows, various participants might want to see a vehicle (i.e. getaway car) as a single object or as a composition of CG Assets (tires, doors, hood). It is important to keep track of individual CG Assets and compositions of CG Assets with enough information to ensure that nothing is forgotten when transferring from one part of the workflow to another and to ensure that the receiver knows what it is getting without having to open and understand each asset and infer or reverse engineer its purpose – there are too many formats and types for an application to know how to deal with them all. This is the case, for example, when CG geometry or whole models come from an external supplier, or components of a CG model are provided by different departments within a production.
- Interoperable data transfer. Given most complex VFX shots require multiple people and organizations, the focus of this version of OMC is for organizations to pass CG assets that are intended to communicate intent (i.e. look of Iron Man's suit) and starting points for more complex processes (i.e. re-topologize a concept model).

These are no different from OMC's treatment of any other Asset type.

These same needs arise as processes that used to be a single pipeline get distributed across an organization or to external organizations, e.g. for managing rendering caches, splitting animation and rendering across different systems, and so on. Future versions of OMC CG Assets will cover some of the more common cases of this.

This effort complements the various efforts from studios in the Open Source community and the Academy Software Foundation to define various standards from scene descriptions (USD) to materials (MaterialX) to geometry (Alembic). While these formats are extensible and can represent complex CG

[1] See https://brightlightsfilm.com/vicious-circle-john-whitney-and-the-military-origins-of-early-cgi/#.ZlX6kXbMKQs

concepts and models, they require the context and connection to the process and creative work that is usually defined as a "pipeline." Given pipelines and processes vary widely between organizations, the CG assets in the OMC sense focus on providing a common description of objects and relationships back to the narrative and production contexts.

1.1 Notational Conventions

In documents generally:

- The definition of a term included in the Dictionary is in bold, followed by the definition, e.g., **Creative Work:** A uniquely identified production.
- When a defined term is used in the text of a document, it is capitalized, for example in "The Production Scene is usually derived from a numbered scene in the Script," Production Scene and Script are defined in the Ontology. (Note, a word that is part of defined term may sometimes be capitalized by itself as a shorthand, e.g., "Scene" may be used to indicate "Narrative or Production Scene.")
- References to other Ontology Documents are in ***bold italic***, e.g., ***Part 3: Assets*** or ***Part 3A: Camera Metadata***

For Sample Attributes in the concept documents:

- If a data field or attribute is formally defined in this ontology or a connected ontology, it is italicized, e.g., *Setup* as an attribute refers to a defined concept.
- Attribute […] indicates an attribute can appear more than once, e.g., *Identifier* […]
- →Thing means that an attribute is expressed as a relationship to a Thing, e.g., the →*Script* attribute of Creative Work means there is a relationship Creative Work→*Script*
- A combination of the two indicates that the concept can have relationships to a set of things, e.g., →Components […]
- Many elements of the Ontology have a Context element. (See **Part 2: Context**.) Relationships declared in the Context are implied to have the item to which the Context is attached as their starting point, for example, Narrative Location→Context→Narrative Scene.

  Contextual relationships that are especially important to the concept being defined are given in the sample attributes tables as C→Thing or C→Thing […] as appropriate. These relationships can just as well be on the object that has the Context. For example, if Narrative Location has "C→Narrative Scene" as an attribute, it is ok to have the relationship directly on the Narrative Location or in its Context, e.g. Narrative Location→Narrative Scene or Narrative Location→Context→Narrative Scene.

  Some implementations (e.g. RDF) place these relationships directly on the class as well as allowing them in Context, and others (e.g. JSON) place all relationship in a Context.

2 CG Assets

This section will grow as the Ontology grows. Obvious candidates for inclusion are structures and formats for times and dates, for example. The intent is to use existing standards and ontologies wherever possible and include them by reference.[2]

Note:

> Although many formats used in computer graphics are interoperable and vendor independent, many are not, or have nuances based on the application that produced them. For this reason, it is good practice to use the "Software" field in Assets and Compositions to provide clarity, e.g. for CG Rigs or for Geometry files that are specific to a particular application or version of an application.

2.1 Geometry

Geometry is a Structural Class (See *Part 3: Assets*.) Its level of granularity covers data exchange and connecting CG Assets to the rest of the workflow, as discussed in the Introduction.

- Information about the actual shape as represented in the Geometry object's associated file (or other way of containing it), such as the type of shape, its orientation, and its bounding box
- information about how it fits in with the rest of the production workflow, such as its scale and abroad indication of its intended purpose.

Geometry: A shape defined in three dimensions

Sample Attributes for Geometry

Term	**Definition**
Identifier […]	One or more identifiers for the Geometry. At least one of these should be resolvable within the production environment.
Name	The name of the Geometry.
Description	A description of the Geometry.
Geometry Type	See below
Bounding Box	See below
Scale	See below
Coordinate Orientation	See below
Purpose	See below
Level of Detail	See below
Geometry File Type	See below

[2] For example, ISO, W3C, and IETF provide many of the things that are needed for times, dates, countries, and languages.

Custom Data	Anything that is application or workflow dependent that can't be otherwise expressed in the Ontology or needs to be present in a particular format.
→ *Context* […]	Any Context for the Geometry. See ***Part 2: Context***

Notes:

Geometry does not have lighting or a camera. In a workflow, those will generally be included in any CG Composition that includes the Geometry Asset. If the bare geometry is imported to an application that assumes the presence of lights and cameras, the application should use its own default values for them.

2.1.1 Geometry Type

There are many different types of geometry files. For example, a surface can be represented as a set of triangles or as a set of mathematical curves of various kinds; a sphere can be represented as a mesh, or parametrically with a center and radius. This type is often independent of the file type, since some file types are used for more than one type of Geometry. Geometry Types indicates the form of the geometry, rather than anything about its use or appearance; individual applications may have to remap or interpret differences between these and their internal formats.

Geometry Type: A description of the general underlying form of a three-dimensional shape.

This table has currently supported OMC Geometry Types. Different tools often support the same conceptual Geometry Type with their own file formats. USD is the emerging standard for these for data interchange, and their definitions are as in USD (see https://openusd.org/release/api/usd_geom_page_front.html)

Geometry Type	***USD Type***	***Notes***
Mathematical Types		
Mesh	UsdGeomMesh	OMC does not distinguish a plain mesh from a subdivision surface. That information is internal to the Geomety file.
NURBS Surface	UsdGeomNurbsPatch	
NURBS Curve	UsdGeomNurbsCurve	
Basis Curve	UsdGeomBasisCurve	OMC does not distinguish B-Splines and Beziers. That information is internal to the Geometry file.
Parametric types		These types provide a compact representation of common shapes. There are many, many others
Cube	UsdGeomCube	
Sphere	UsdGeomSPhere	
Cylinder	UsdGeomCylinder	

Cone	UsdGeomCone	
Torus		
Plane		USD plans to add this.
Capsule	UsdGeomCapsule	
Teapot		Included for reasons of nostalgia.

2.1.2 Bounding Box

It is often convenient to know approximately how big a piece of geometry is. This can be specified in various ways – convex hull, bounding sphere, etc. OMC uses the simplest form.

Bounding Box: The minimum axis-aligned right rectangular prism[3] in the local space of the Geometry that fully encloses the Geometry.

The Bounding Box is specified by two diagonally opposite points.

Attributes for Bounding Box

Term	**Definition**
Corner 1	One corner of the bounding box, given as a *Point3*. See ***Part 9: Utilities***
Corner 2	The diagonally opposite corner to Corner 1, given as a *Point3*. ***See Part 9: Utilities***

Notes:

A Point3 is a triple of numbers, in this case denoting x, y, and z.

Other bounds-related values, such as the maximum value in any direction and the footprint in a dimension, can be computed from the Bounding Box.

2.1.3 Scale

Much of the workflow downstream from creating the geometry depends on scale. Ensuring that everything matches appropriately simplifies rigging, simulations, textures, and much more: the boat has to fit the dock, the people have to fit in the boat, and the boat's flag really shouldn't be bigger than the boat. Otherwise, the rendering pipeline can have the same problem as the G'Gugvuntts:

> …the mighty ships tore across the empty wastes of space and finally dived screaming on to the first planet they came across - which happened to be the Earth - where due to a terrible miscalculation of scale the entire battle fleet was accidentally swallowed by a small dog.
>
> - *The Hitchhiker's Guide to the Galaxy* (1979)

[3] Also known as a rectangular cuboid.

Scale: The number of "real" units represented by a single unit in the coordinate space of the Geometry.

Note:

This is expressed as a Measurement (see ***Part 9: Utilities***.)

USD constrains this to "meters per unit".[4]

Scale is also used for other CG Assets and CG Compositions.

2.1.4 Coordinate Orientation

Coordinate Orientation describes the relationships of the coordinate axes to each other. In the origins of computer graphics, the traditional (x,y) CAD plotting system used z to denote height. Video games later used z for depth as the TV screen represented x and y and depth was the distance from that screen plane.

Note that not all possible orientations are supported. OMC assumes the positive X-axis is always to the right, and that the second axis (Y or Z) is always "up". This reduces the possible combinations to four.

In current practice:

	Left-handed	**Right-handed**
Y-up	Unity LightWave ZBrush Maxon Cinema4D	Maya Houdini Godot game engine Minecraft USD (configurable) glTF
Z-up	Unreal	3DS Max Blender Autocad USD (configurable)

Coordinate Orientation: The direction and handedness of the axes used in the geometry.

Sample Attributes for Coordinate Orientation

Term	**Definition**
Up Axis	String. Allowed values are "Y-up" and "Z-up"
Handedness	String. The handedness of the third axis. Allowed values are "left" and "right"

[4] Since "scale" can have other meanings in some CG pipelines this is sometimes called "units." However, "unit" is used in OMC as an attribute of a Measurement.

Notes:

USD allows Y-Up and Z-up, and is always right-handed.

glTF is Y-up and righthanded.

Applications that have a left-handed third axis, such as Unity and Unreal, can convert the geometric data as necessary, as can Y-up applications that are given Z-up data (and vice versa.)

2.1.5 Purpose

For Geometry, Purpose is how the Geometry itself is intended to be used when rendering *as geometry,* e.g. for positioning reference, as a lower-resolution proxy for pre-visualization or rapid playback, for final rendering, etc. This is distinct from its use with an Asset Functional Class, which details how the Geometry connects to the production, e.g. as a Prop or a Character.

This is broader than USD's Imageable Purpose field.

Purpose: A suggested or intended use for the object in a pipeline.

Here are some examples of Purpose. Implementations should use these when possible. This list will be extended over time.

Type	Domain	Description
Rendering	Film/Real-Time/Web	Final high-quality visual output of 3D geometry.
Proxy	Film/Games	Lower-resolution geometry used for faster pre-visualization and playback.
Collision	Film/Games/Real-Time	Simplified geometry for collision detection and physics simulations.
Guide	Film	Geometry used as a visual guide or reference in animation and VFX.
Matte/Paint	Film	Geometry that acts as a background or underlay for visual atmosphere creation.
Annotation	Multiple	3D elements that provide additional information or commentary on other geometry.
General	Multiple	
Printing	Multiple	Geometry built as input for a 3D printer process

Notes:

Purpose is also used for other CG Assets and CG Compositions.

2.1.6 Level of Detail

Purpose is often mixed with some notion of level of detail, such as lo-res or high-res. Level of detail is always subjective, and is independent of the Purpose and Scale of the Geometry. This value is often derived from the production planning process where the task to create the CG asset has information on

whether the object will be in the background, midground or foreground relative to the virtual camera in one or more VFX or animation shots. This information combined with the image resolution of the distribution format of the creative work (i.e. 4K vs HD) will guide the artist in terms of geometry and texture resolution. This attribute will help participants understand the intended level of detail of a CG asset at the time of creation.

Level of Detail: Percentage of the screen that an object can reasonably take up.

For example, if a starship appears in the far background, it does not have to be an extremely detailed model – in this case, the level of detail would be low, e.g. 5%. If it appears in the foreground, more detail is required, so the object looks good when it takes up 80% of the screen. A close-up tree needs every single leaf done out; at medium scale a leaf may be two textured triangles; at a far distance, a tree can be a plane with a texture on it.

Another example is to create a high resolution model with a high level-of-detail via sculpting, photogrammetry, proceduralism and then programmatically derive several CG assets each with a lower of detail through voxel based geometry reduction methods. Each derived CG asset would have a lower level of detail value. Textures can also be re-projected onto the new geometry with lower quality and size.

Level of Detail is also used for other CG Assets and Compositions.

Notes:

> Level of Detail in a given production is directly related to the creative work master distribution format (i.e. 4k vs. HD). Therefore, when archiving CG assets or creating a library these values can inform future use. For example, an object with a high level of detail in a creative work which was created with HD as the master format, might be a background object in a creative work which is aiming to be in 4k. This was masterfully executed in the Toy Story 4 narrative location “Second Chance Antiques” which created a cinematic background of old toys using several CG assets from previous Toy Story films that were used in close-up animation shots.

2.1.7 Geometry File Type

The following is a list of common industry file types for CG Assets. For OMC’s use of the word “file” for general external data and the use of these as OMC file types, see ***Part 9: Utilities*** under “File details”.

Name	File Extension	Description
USD	.usd .usda .usdc .usdz	Universal Scene Description, for large-scale 3D scenes.
FBX	.fbx	A 3D data exchange format that stores 3D assets, characters and animation.
Alembic	.abc	A 3D interchange format for animated geometry.
OBJ	.obj	Legacy Alias file format used for interchange.
GlTF	.gltf .glb	A JSON-based format for efficient delivery of 3D content.
Cinema 4D	.c4d	Native format for Cinema 4D.

Name	File Extension	Description
Blender	.blend	Native format for Blender
Maya	.ma, .mb	Native format for Maya
Houdini	.geo	Native format for Houdini

2.2 Volume

Geometry is one way of defining things in 3D space, and focuses mostly on the surfaces of objects. Objects in 3D space can also be described as combinations of volumes, and, like geometry, can be done with coarse or fine approximations of the object in question.

Volume: A non-mesh representation of 3D data

Volume is an Asset Structural Class.

Sample Attributes for Volume

Term	Definition
Identifier […]	One or more identifiers for the Volume. At least one of these should be resolvable within the production environment.
Name	The name of the Volume.
Description	A description of the Volume.
Bounding Box	As for Geometry
Scale	As for Geometry
Coordinate Orientation	As for geometry
Axis Aligned	True if the volume is aligned with the three axes of the coordinate space.
Level of Detail	As for Geometry, when it is relevant to the use of the Volume.
Volume File Type	See below
Custom Data	Anything that is application or workflow dependent that can't be otherwise expressed in the Ontology or needs to be present in a particular format.
→ *Context* […]	Any Context for the Geometry. See ***Part 2: Context***

Notes:

Volumes can be created by scanning, simulations, desktop tools, and processing of scanned data. Procedural Volumes will be covered in a future release of the Ontology.

2.2.1 CG Volume

There are many uses for Volumes in computer graphics. There are other uses as well, which will be covered in future releases of other parts of the Ontology.

CG Volume: A Volume used in a CG pipeline.

CG Volume is an Asset Functional Class.

Sample Attributes for CG Volume

Term	Definition
Identifier […]	One or more identifiers for the CG Volume. At least one of these should be resolvable within the production environment.
Name	The name of the CG Volume.
Description	A description of the CG Volume.
CG Volume Purpose	See below
Custom Data	Anything that is application or workflow dependent that can't be otherwise expressed in the Ontology or needs to be present in a particular format.
→ *Context* […]	Any Context for the Geometry. See ***Part 2: Context***

Notes:

2.2.2 Volume File Types

VDB (and openVDB, its standard implementation) is becoming common, but many 3D modelling and rendering applications have their own types as well. For OMC's use of the word "file" for general external data and the use of these as OMC file types, see ***Part 9: Utilities*** under "File details".

Name	File Extension	Description
VDB	.vdb	Open source Standard for volumes (smoke, fire, fluids, clouds)
Field3D	.f3d	Legacy volumetric format (replaced by OpenVDB)
Houdini BGEO	.bgeo	Houdini's native volume/FX caches
Bifrost	.bif	Autodesk native volume format

2.2.3 CG Volume Purpose

As with Materials and CG Rigs, terminology and uses for CG Volumes are proliferating and non-standard. This list of CG Volume Types provides a set o examples, not a normative list. Implementations may decide to implement these as subclasses of CG Volume rather than as a property value.

CG Volume Purpose: The intended use of a CG Volume in a CG pipeline or system.

CG Volume Purpose	Description	Examples
Collision	Volumes used as collision fields in simulations (fluids, smoke, rigid bodies). These define where particles/fluids interact with solid objects.	OpenVDB collision volumes (.vdb from Houdini or Bifrost), Houdini .bgeo SDFs, Maya Bifrost .bif collision meshes
Light Transport	Volumes that define how light scatters and absorbs within media (fog, smoke, fire). Used in rendering and lookdev.	OpenVDB density/grids (.vdb), Deep OpenEXR .exr (for rendered fog/smoke), Arnold/V-Ray volumetric shaders
Surface Effect	Volumes used to drive secondary effects on surfaces like foam, mist, erosion, or displacement. They modify geometry indirectly.	Bifrost foam/mist caches (.bif), Houdini .bgeo surfacing fields, OpenVDB curvature/velocity fields
Rendering	Volumetric data directly consumed by render engines to produce final pixels (smoke, fire, clouds, explosions).	OpenVDB (.vdb) rendered in Arnold/RenderMan/Redshift, Deep EXR passes for comp (.exr), Alembic Volumes .abc
Other	Miscellaneous uses of volumes such as scientific sims, voxel art, game engines, and research.	HDF5 `.h5` simulation grids, MagicaVoxel `.vox`, DDS 3D textures `.dds`, Field3D `.f3d` (legacy)

2.3 Point Cloud

Unlike Geometry, which describes edges and surfaces, and volumes, which describe regions of 3D space, point clouds describe discrete points. Point clouds can be captured or generated, but the underlying structural class for the data is the same for both.

Point Cloud: A set of data points in 3D space

Sample Attributes for Point Cloud

Term	Definition

Identifier […]	One or more identifiers for the Point Cloud. At least one of these should be resolvable within the production environment.
Name	The name of the Point Cloud.
Description	A description of the Point Cloud.
Bounding Box	As for Geometry
Scale	As for Geometry
Coordinate Orientation	As for geometry
Point Cloud File Type	See below
Num Points	Optional Integer. The number of individual points in the Point Cloud.
Custom Data	Anything that is application or workflow dependent that can't be otherwise expressed in the Ontology or needs to be present in a particular format.
→ *Context* […]	Any Context for the Geometry. See ***Part 2: Context***

Notes:

Point Clouds are often used to generate other Assets, such as deriving Geometry from a set of scanned points. These new Assets should use OMC Derivations to connect to the underlying Point Cloud.

2.3.1 CG Point Cloud

There are many uses for Point Clouds in computer graphics. There are other uses as well, which will be covered in future releases of other parts of the Ontology.

CG Point Cloud: A Point Cloud used in a CG pipeline.

CG Point Cloud is an Asset Functional Class.

Sample Attributes for CG Point Cloud

Term	**Definition**
Identifier […]	One or more identifiers for the CG Point Cloud. At least one of these should be resolvable within the production environment.
Name	The name of the CG Point Cloud.
Description	A description of the CG Point Cloud.
CG Point Cloud Purpose	See below
Custom Data	Anything that is application or workflow dependent that can't be otherwise expressed in the Ontology or needs to be present in a particular format.

→ *Context* […]	Any Context for the Geometry. See ***Part 2: Context***

Notes:

2.3.2 Point Cloud File Type

LAS is a well-known and standard format for point cloud data, as is OpenParticleIO. Many applications support proprietary formats. For OMC's use of the word "file" for general external data and the use of these as OMC file types, see ***Part 9: Utilities*** under "File details".

Name	File Extension	Description
LAS	.las	Open standard format for point cloud data, commonly used in LiDAR scanning.
LAZ	.laz	Lossless compressed form of LAS; widely used to reduce storage size while preserving data.
OpenParticleIO (OpenPNT/OPIO)	.pio	Open format for particle/point cloud interchange in VFX and simulation workflows.
Alembic	.abc	Industry-standard cache format; supports geometry, particles, and point cloud data for interchange between DCC apps.
PLY	.ply	Polygon/point cloud format often used for 3D scanning data and geometry interchange.
XYZ	.xyz	Simple ASCII format for point cloud coordinates (common in LiDAR/scanning).

2.3.3 CG Point Cloud Purpose

CG Point Cloud Purpose: The intended use of a CG Point Cloud in a CG pipeline or system.

CG Point Cloud Purpose	Description	Examples
Particle	A point cloud representing particles in motion, used to simulate atmospheric and fluid effects such as smoke, mist, sparks, fire embers, or bubbles.	Houdini particle caches (.bgeo), Bifrost particles (.bif), OpenParticleIO caches
Geometry Source	A point cloud used as the basis for geometry generation — either meshing particles into	Meshing liquid sims into water surfaces, scattering trees on terrain, crowd/prop instancing

	surfaces or scattering geometry instances onto points.	
Lighting / Shading Data	A point cloud used as an input field or driver for simulations (fluid sources, velocity fields, density emitters).	Emitter point clouds in Houdini/Maya; LiDAR scans used as collision or density fields
Scanning / Capture	A point cloud derived from LiDAR, photogrammetry, or 3D scanning, used as reference or source data in the pipeline.	.las, .laz, .e57, .ply — used in VFX environments, digital doubles, set reconstruction
Visualization	A point cloud rendered directly for data visualization or stylized effects, not meshed into surfaces.	Scientific visualizaton of particle simulations, stylized "point cloud look" in music videos, VR previews

Notes:

Formalization of NERFs and Splats is deferred to a future release.

2.4 Map

When rendered, plain geometry is not very realistic or convincing – the results may have colors and some effects from lighting, but that's all. Texture mapping is the umbrella term for a set of techniques to add detail, texture, and color, originally proposed by Catmull in his doctoral dissertation, "A subdivision algorithm for computer display of curved surfaces" (1974). Maps now cover much more than traditional textures.

Maps are based on Digital Images (See ***Part 3F: Images***.) Occasionally they are procedural Assets (See **Part 3: Assets**.)

Map: An image intended to drive or influence a behavior or value within a CG workflow.

For example, Maps can be used for color, deformation, describing no-go area in games, and much else.

The use of Maps continues to evolve, so proposed field values in this section are not exhaustive.

Sample Attributes for Map

Term	**Definition**
Identifier […]	One or more identifiers for the Map. At least one of these should be resolvable within the production environment.
Name	The name of the Map.
Description	A description of the Map.
Map Type	See below.
Map Format	See below.

Term	Definition
Scale	Optional. If not present, it is inferred base on the Geometry the Map is intended to be used with.
Level of Detail	See above
Custom Data	Anything that is application or workflow dependent that can't be otherwise expressed in the Ontology or needs to be present in a particular format.
→ *Context* […]	Any Context for the Asset. See ***Part 2: Context***

Notes:

A Map is an Asset Functional Class. Its Structural Class is usually a Digital Image or Asset Group (see UDIM, below), though it can be a Procedural Asset.

Information about the Map's file type is kept with the Asset Structural Class.

2.4.1 UDIM

UV Maps provide one texture for an entire model, which means that the resolution of the Map is the same everywhere on the model, even if some areas need more detail, which can result in a very large map. Conversely, a lower resolution Map will not be detailed enough for parts of the model. Most systems support UDIMs (U Dimension textures) to ameliorate this problem by diving the Map into tiles that span the entire UV space for the model. UDIMs also make it easier to create, modify, and use individual portions of the map.

The pieces of a UDIM are called tiles. Each tile is numbered with its position in the UV space covered by the Map. Tiles are identified by four-digit numbers starting at **1001**, laid out in 10 columns across **U** (0–9) and unlimited rows in **V**. The tile index is computed as 1001 + u + 10*v. This information is usually carried in the file name of the tile.

UDIM Tile is a Functional Class for Digital Image structural classes. In OMC, a Map implemented as a UDIM is like any other Map, with the exception that its Asset Structural Class is an Asset Group containing only UDIM Tiles.

UDIM Tile: A numbered sub-piece of a UV Map.

Sample Attributes for UDIM Tile

Term	Definition
Identifier […]	One or more identifiers for the UDIM Tile. At least one of these should be resolvable within the production environment.
Name	The name of the UDIM Tile.
Description	An optional description of the UDIM Tile.

Term	Definition
UDIM Tile Number	Integer
Custom Data	Anything that is application or workflow dependent that can't be otherwise expressed in the Ontology or needs to be present in a particular format.
→ *Context* […]	Any Context for the Asset. See ***Part 2: Context***

Notes:

The UDIM Tile Number should match the standard tile number suffix that is usually embedded in a file name.

The resolution (dimensions) of a UDIM tile can be found in its Structural Characteristics.

2.4.2 Map Type

Different types of Map influence rendering in different ways. There are many types of Map, some historical, some cutting edge, and some old standbys. Because of this variety and changeability, map type is implemented as a property rather than a subtype. We recommend using map types from appropriate standards and specifications.

Map Type: Guidance about the intended use of a Map in a Material.

The following are taken from the Open PBR ASWF Project (https://github.com/AcademySoftwareFoundation/OpenPBR):

- Diffuse
- Specular
- Metalness
- Roughness
- Opacity
- Ambient Occlusion
- Normal
- Bump

These are taken from current and historical[5] industry practices:

- UV Map
- Projection Map
- Height Map

[5] Especially useful when puling things from an archive.

- Weight Map
- Depth Map
- Albedo

2.4.3 Map Format

Map Format: The data layout of a Map.

As with Map Type, Map Format has only partial industry convergence. OIIO calls this Texture Type (https://openimageio.readthedocs.io/en/latest/stdmetadata.html) and is one of the sources for this list.

Format Name	Format Description
Plain Texture	Ordinary 2D texture
Volume Texture	3D volumetric texture
Shadow	Ordinary *z*-depth shadow map
CubeFace Shadow	Cube-face shadow map
Volume Shadow	Volumetric (“deep”) shadow map
LatLong Environment	Latitude-longitude (rectangular) environment map
CubeFace Environment	Cube-face environment map
Emissive	Specifies how much light is emitted from each point in the Map
Emissive Color	Specifies how colored light is emitted from each point in the Map

2.5 Material

Texture only provides some information to a renderer. Material provides further details, realism, or artistic effects and controls the final appearance of the CG model. Materials can be simple – a single file – or complex – a Composition that includes other elements that contain information about reflectivity, detailed surface properties, and color. Complex materials can be ‘baked’ into a static representation for export or performance.

Materials come in a variety of standardized and proprietary formats, such MaterialX.

Materials can use multiple textures, or none, and can refer to other Materials. They can also contain Procedural Assets.

Some Materials are explicitly related to a particular type of surface. This is often true of libraries of standard base materials, to make searching and filtering easier – an application can present a user with

only “stone” or “wood”, rather than anything else. These types are not currently standardized, but see below for some general suggestions.

In OMC, Materials, which are Assets, and Composite Materials, which are Compositions, can use all the usual mechanisms for Versions. (See ***Part 3B: Versions***.)

2.5.1 Material

Material: Data values and relationships required to describe the look of a CG Asset

Sample Attributes for Material

Term	Definition
Identifier […]	One or more identifiers for the Material. At least one of these should be resolvable within the production environment.
Name	The name of the Material.
Description	A description of the Material.
isSelfContained	True if the Material’s file has no external references; false or absent otherwise. See ***Part 9: Utilities***
Material Type	See below.
Purpose	Optional. See above.
Level of Detail	Optional. See above.
Scale	Optional. See above.
Custom Data	Anything that is application or workflow dependent that can’t be otherwise expressed in the Ontology or needs to be present in a particular format.
→ *Context* […]	Any Context for the Simple Material. See ***Part 2: Context***

Notes:

A Material is an Asset Functional Class. Its structural portion can be digital data, a digital document or, in some cases, an Image or Procedural Asset.

Information about a Material’s file type is kept with its Asset Structural Class.

Purpose, Level of Detail, and Scale can be included to prevent mismatches between a Material and the Geometry to which it is applied.

Material files that have external references to maps, sub-materials, etc. should have isSelfContained set to false or absent.

2.5.2 Material File Type

Many material formats are proprietary but there are some emerging open standards like MaterialX. For OMC's use of the word "file" for general external data and the use of these as OMC file types, see ***Part 9: Utilities*** under "File details".

Name	File Extension	Description
Materialx	.mtlx	A collaborative, open-source standard from ILM for exchanging rich material and look-development content across different software.
OpenUSD	.usd	OpenUSD is a collaborative, open-source framework from Pixar for handling 3D scene data, including geometry and materials.
Substance	.sbsar	This proprietary format is produced by the Adobe Substance 3D tools and can define procedural materials with a high degree of flexibility.
Material Template Library	.mtl	A companion material library file for Wavefront OBJ models; defines surface appearance properties (colors, shininess, texture maps) for 3D objects
FBX	.fbx	A proprietary Autodesk format for exchanging complete 3D scenes; includes geometry, animation, materials, and shader assignments.

2.5.3 Shader

Shaders are general purpose building blocks that can be used to achieve many different looks. These looks can be controlled in several ways:

- If the Shader is a file, e.g., OpenPBR, the Shader's Asset Structural Class is represented as Digital Data.
- If the Shader is implemented with code inside a DCC or renderer, its Asset Structural Class is a Procedural Asset, and the details (software used, configuration, parameters, and so on) are described by the Software field of the Procedural Asset.

A Shader is a subclass of Material.

Shader: A programmable or software-defined Material.

Notes:

2.5.4 Shader File Type

Many shaders formats are proprietary but there are some emerging open standards like OpenPBR. For OMC's use of the word "file" for general external data and the use of these as OMC file types, see ***Part 9: Utilities*** under "File details".

Name	File Extension	Description
OpenPBR	.mtlx or .usd	An open, physically based *material model* defined in MaterialX that standardizes parameters and behaviors so materials look consistent across DCCs and renderers.
Material Definition Language	.mdl	A declarative language from Nvidia that describes *what* a material is (not *how* to shade it) so the same definition renders consistently across MDL-capable tools.
HLSL	.hlsl	Microsoft's real-time shader language for Direct3D pipelines, compiled to GPU bytecode for programmable stages (vertex, pixel, geometry, hull, domain, compute).
GLSL	.vert/.frag/.geom	A cross-platform, C-like real-time shader language for OpenGL/Vulkan pipelines used to implement programmable stages on the GPU.
Open Shading Language	.osl	Open-source shading language by Sony Imageworks for writing renderer-independent shader code; widely supported in modern renderers (Arnold, V-Ray, Octane, Blender Cycles, etc.) to create custom materials

2.5.5 Composite Material

More complex Materials are Compositions: to do their job, they require many individual components, such as Maps and sub-Materials, which are then combined.

Composite Material: A material represented by a Composition.

Sample Attributes for Composite Material

Term	Definition
Identifier […]	One or more identifiers for the Composite Material. At least one of these should be resolvable within the production environment.
Name	The name of the Composite Material.
Description	A description of the Composite Material.
Material Type	See below.
Purpose	Optional. See above.

Level of Detail	Optional. See above.
Scale	Optional. See above.
Asset ->[] *Asset Structural Class*->[] *Composition*->[…]	Inherited from Composition; use the appropriate one for any inclusions that don't fall into the other categories.
->startHere	Material. The Material (an Asset) used as the starting point for using this Composite Material. Inherited from Composition.
->*Map*[…]	Any Maps used directly in this Composite Material.
-> *Material*[…]	Any Materials used directly in this Composite Material.
-> *Composite Material*[…]	Any Composite Materials used directly in this Composite Material.
Custom Data	Anything that is application or workflow dependent that can't be otherwise expressed in the Ontology or needs to be present in a particular format.
→ *Context* […]	Any Context for the Simple Material. See ***Part 2: Context***

Notes:

A Composite Material is a Composition. See ***Part 9: Utilities*** for possible implementation strategies.

Included Assets might include, for example, any Procedural Assets needed by the Composite Material.

The Map, Material, and Composite Material fields are expected to be the external references used in the startHere Asset.

The relationships for the included elements are includesMaterial/includedBy, includesMap/includedBy, and includesCompositeMaterial/includedBy.

Information about the Composite Material's file type is kept with the Asset Structural Class.

Purpose, Level of Detail, and Scale can be included to prevent mismatches between a Composite Material and the Geometry to which it is applied.

2.5.6 Material Type

As mentioned, when a workflow has many – hundreds or even thousands – of Materials, there is usually some way of categorizing them. This is not in any way standardized, and we hope that as more systems use OMC, some practices will emerge.

Material Type: Guidance about the intended use or effect of a Material.

It is often convenient to group Materials into broad top-level categories, with an additional level of refinement. Although an implementation can do this with subclassing for the top level type and further subclassing for more detailed types, OMC prefers a *type.subtype* value in the Material Type field; this is easier to manage while a thousand flowers are blooming and practices are local and insular.

For example, this set of values approximates some relatively common practices[6].

Material Type	**Material Subtype**	**Final value**	**Notes**
Organic		Organic	
	Plant	Organic.plant	
	Skin	Organic.skin	
	Hair	Organic.hair	
	Fur	Organic.fur	
	Scales	Organic.scales	
	Leather	Organic.leather	
	Wood	Organic.wood	May have further subclassing based on the kind of wood, how it is sawn and sanded, etc.
Textile		Textile	
	Cotton	Textile.cotton	
	Silk	Textile.silk	
	Wool	Textile.wool	May be subclassed for twill, tweed, serge, etc.
Inorganic		Inorganic	
	Plastic	Inorganic.plastic	
	Metal	inorganic.metal	
	Glass	Inorganic.glass	
	Stone	Inorganic.stone	May be subclassed polished rough granite, smooth granite, sandstone, etc.
GasOrLiquid		GasOrLiquid	
	Atmosphere	GasOrLiquid.atmosphere	
	Fog	GasOrLiquid.fog	
	Water	GasOrLiquid.water	
	Oil	GasOrLiquid.fire	
	Fire	GasorLiquid.oil	With subclasses for olive, crude, and Abramelin.

[6] Note that even with this, it is unclear where to put a glass brick as seen in New York and London sidewalks and in many Streamline Moderne buildings. Blood has a similar problem – Organic or Liquid? This shows why this has been a standards-resistant topic.

2.6 CG Rig

CG Rig: An Asset that acts a control system for another entity.

The current version of OMC only applies rigs to CG Assets and CG Assemblies. Rigs can also be applied to CG lights and cameras, which will be addressed in a future release.

Sample Attributes for CG Rig

Attribute	**Description**
Name	The name of the CG Rig
Identifier […]	One or more identifiers for the CG Rig. At least one of these should be resolvable within the production environment; others might point to sources with more information.
Description	A description of the CG Rig
Bounding Box	As for geometry – see above.
Scale	As for geometry – see above.
Coordinate Orientation	As for geometry – see above.
CG Rig Type	The intended use of a CG Rig. See below.
Version	See **Part 3B: Versions**
Custom Data	Anything that is application or workflow dependent that can't be otherwise expressed in the Ontology or needs to be present in a particular format.
→ *Context*	Context related to the CG Model.

Notes:

This is a Functional class for use with Digital Data or a Procedural Asset.

There is little standardization around CG Rigs, so no internal detail is provided.

CG Rigs can be derived from other CG Rigs.

"Armature" is currently used very loosely. In some environments, an Armature can be more portable than a CG Rig, but this is not always the case. A future version of OMC will re-examine Armatures.

2.6.1 CG Rig Type

As with Materials, there are many types of CG Rigs and local practices will have some way of categorizing them. In the absence of standards for CG Rigs, OMC provides some broad top level categories. Again like Materials, and implementation can use these to create subclasses, but given the complexity of CG Rigs in general, using the CG Rig Type field is preferable for now. Subtypes can be described using standard `type.subtype` notation for the value of the field, e.g. "`quadruped.winged`"

CG Rig Type: Guidance about the intended use of a CG Rig.

These categories are very, very broad, and will be most useful with workflow-specific subtypes.

CG Rig Type	Description	Examples
Biped	A CG Rig for something that walks on two feet	A person or an elf
Quadruped	A CG Rig for something that walks on four feet	A horse or a badger
Polyped	A CG Rig for something that walks on more than four feet	A spider or a crab
Avian	A CG Rig for something whose motion is birdlike	A sparrow or an eagle
Serpentine	A CG Rig for something that slithers.	A snake or an eel; possibly a squid.
Appendage	A CG Rig for a separately controlled extension	An arm, a tail, the digging attachment on a backhoe, a tongue.

2.7 CG Model

Although many CG Assets can be treated functionally as Production Props, Production Set Dressing, and so on, there are times when the purpose of a CG Asset is not known, or known only loosely, or when included in a Geometry or CG Assembly. Many parts of the Ontology need a full Asset with functional and structural components rather than a bare structural element. To cover these cases, OMC has a functional class for CG assets whose purpose is ambiguous or unknown.

The isSelfContained property is especially useful for these, since it means that the Asset and its structural component can be used independently of other assets, for example a USDZ file or a self-contained glTF or GLB file.

CG Model: A generic use for Geometry

Sample Attributes for CG Model

Attribute	**Description**

Name	The name of the CG Model
Identifier […]	One or more identifiers for the CG Model. At least one of these should be resolvable within the production environment; others might point to sources with more information.
Description	A description of the CG Model, e.g. “silver French horn”
isSelfContained	True if the CG Model’s structural component has no external references; false or absent otherwise. See ***Part 9: Utilities***
Version	See **Part 3B: Versions**
Custom Data	Anything that is application or workflow dependent that can’t be otherwise expressed in the Ontology or needs to be present in a particular format.
→ *Context*	Context related to the CG Model.

Notes:

CG Model is a subclass of Production Object

A CG Model might not have any information about how it is used in the production process, for example when doing simple asset exchange. That information can be added once it is known, or the CG Model’s structural element can be connected to another production-specific Asset, such as a Production Prop.

A CG Model can be used as the startHere property of a CG Assembly or a Geometry Assembly.

2.8 CG Assembly

A CG Assembly is a subclass of a Composition (See ***Part 9: Utilities***.) It is not itself an Asset, but a collection of Assets that can be used to generate other Assets, such as a turntable or some video.

A CG Assembly groups together everything needed to use a model, especially to render it. It can include CG Geometry and other CG Assemblies, Lights and Cameras, and any other required Assets, such as Maps, Materials, and other data such as instructions, parameter files, and so on.

CG Assembly: A Composition that includes a collection of related CG Assets and CG Assemblies

CG Assemblies can describe individual components - everything needed to render a visually complete Portrayal of a Character or Depiction of a Prop, or a fully decorated house, for example. They can also represent completed portions of a scene (a villager holding a pitchfork with rigging to wave it around) or entire scenes (the character, the mob of angry villagers, the house, and the woods around the house.)

CG Assemblies are a logical overlay on top of any underlying arrangement of the Assets, e.g. a folder and file hierarchy. This enables better re-use of shared components, such as Materials, without being tied to a particular application specific arrangement.

Sample Attributes for CG Assembly

Term	Definition
Identifier […]	One or more identifiers for the CG Assembly. At least one of these should be resolvable within the production environment.
Name	The name of the CG Assembly.
Description	A description of the CG Assembly.
->startHere	Inherited from Composition. The Asset to use as the starting point for traversing this CG Assembly.
Asset ->[] *Asset Structural Class*->[] *Composition*->[…]	Inherited from Composition; use the appropriate one for any inclusions that don't fall into the other categories. This allows subclasses of Asset and Composition, the types of which can be found by inspection of the data.
->*Map*[…]	Any Maps used directly in this CG Assembly. The relationship is includesMap/includedBy.
->*Material*[…]	Any Materials used directly in this CG Assembly. The relationship is includesMaterial/includedBy.
-> *Composite Material*[…]	Any Composite Materials used directly in this CG Assembly. The relationship is includesCompositeMaterial/includedBy.
->*Geometry*[…]	Any Geometry used directly by this CG Assembly. The relationship is includesGeometry/includedBy.
->*Geometry Assembly*[…]	Any Geometry Assemblies used directly by this CG Assembly. The relationship is includesGeometryAssembly/includedBy.
-> CG *Rig* []	Any CG Rig needed for the CG Assembly
->CG *Assembly*[…]	Any CG Assemblies used directly by this CG Assembly. The relationship is includesCGAssembly/includedBy.
Purpose	Optional.
Level of Detail	Optional.
Scale	Optional.
->Light Info […]	Light information for the CG Assembly NOTE: if no light, applications importing OMC data should use their default, what they find reasonable, etc.
->Camera Info	Camera information for the CG Assembly. NOTE: if no light, applications importing OMC data should use their default, what they find reasonable, etc.

Custom Data	Anything that is application or workflow dependent that can't be otherwise expressed in the Ontology or needs to be present in a particular format.
→ *Context* […]	Any Context for the Asset. See ***Part 2: Context***

Notes:

If the CG Composition is imported to an application that assumes the presence of lights and cameras, the application should use its own default values for them. If the CG Assembly is included in a containing CG Assembly, it can inherit lights and cameras if no others are present.

Elements of the Composition that are not explicitly mentioned above can be included in the CG Assembly's field for included Assets and Compositions.

Implementations can use CG Assembly completely generically by excluding all of the specialized fields and including only the fields inherited from Composition.

Light and Camera information are not defined in OMC 2.6 and are included as placeholders.

2.9 Geometry Assembly

Geometry is often created before or in parallel with Maps, CG Rigs, and Materials. In addition, it can be useful to have Geometry for Props items of Set Dressing to be "skinned" in many different ways, in the same or other productions.

These geometry-only models can be composed of multiple pieces, e.g. a tool with different handles or a character whose head and torso are created by different artists. OMC supports these workflows with a Geometry Assembly – like a CG Assembly but containing only geometry information.

Geometry Assembly: A Composition containing only Geometry information.

Sample Attributes for Geometry Assembly

Term	**Definition**
Identifier […]	One or more identifiers for the Geometry Assembly. At least one of these should be resolvable within the production environment.
Name	The name of the Geometry Assembly.
Description	A description of the Geometry Assembly.
->startHere	Geometry Asset. The starting point for this traversing this Geometry Assembly.
Asset ->[] *Asset Structural Class*->[] *Composition*->[…]	Inherited from Composition. NOTE: Since Geometry Assembly includes only geometry, these should not be used other than in exceptional circumstances.

Term	Definition
->*Geometry*[…]	Any Geometry used directly by this Geometry Assembly. The relationship is includesGeometry/includedBy.
->*Geometry Assembly*[…]	Any Geometry Assemblies used directly by this Geometry Assembly. The relationship is includesGeometryAssembly/includedBy.
Purpose	Optional.
Level of Detail	Optional.
Scale	Optional.
Custom Data	Anything that is application or workflow dependent that can't be otherwise expressed in the Ontology or needs to be present in a particular format.
→ *Context* […]	Any Context for the Asset. See ***Part 2: Context***

Notes:

This is included to support geometry-only workflows. It is possible to use a CG Assembly with only Geometry fields instead. It also simplifies management of Variants where the only changes are in Materials and Maps – each Variant refers to the same Geometry Assembly and its own Materials and Maps (e.g. separate ones for dry, wet, and dirty clothing.)

Appendix A External Definitions

These are terms defined elsewhere in the Production Ontology, included here for ease of reference.

Media Creation Context: Informs scope within the construction process of a Creative Work.

See **Part 2: Context**

Asset: A physical or digital object or collection of objects specific to the creation of the Creative Work.

See **Part 3: Assets**

Camera Metadata: Capture-specific details and information about the Camera itself.

See **Part 3A: Camera Metadata**

Participant: The entities (people, organizations, or services) that are responsible for the production of the Creative Work.

See **Part 4: Participants**

Task: A piece of work to be done and completed as a step in the production process.

See **Part 5: Tasks**

Creative Work: A uniquely identified production.

See **Part 6: Creative Works**

Relationship: Describes and defines the connections between elements of the Ontology, such as Assets, Tasks, Participants, and Contexts.

See **Part 7: Relationships**

Infrastructure: The underlying systems and framework required for the production of the Creative Work; it is generally not specific to a particular Creative Work.

See **Part 8: Infrastructure**

Utilities: Common data models and data structures used in multiple places and in multiple ways in a larger system.

See **Part 9: Utilities**

Identifier: An identifier uniquely identifies an entity within a particular scope.

See **Part 9: Utilities**