# An Extension of glTF for Robot Description

Ryoya Izawa[1] and Masanobu Koga[2*]

[1]Department of Creative Informatics, Kyushu Institute of Technology,
Fukuoka, Japan (izawa@mk.ces.kyutech.ac.jp)
[2]Department of Intelligent and Control Systems, Kyushu Institute of Technology,
Fukuoka, Japan (koga@ces.kyutech.ac.jp) * Corresponding author

**Abstract:** In this paper, we propose a new 3D file format RDTF, which is compatible with glTF and can be used for robots simulation. Since RDTF is an extension of glTF, RDTF file can be used with tools that support glTF. It is worth mentioning that there are many existing glTF models[3] which can be transformed to RDTF.

**Keywords:** Computer Graphics, glTF

## 1. INTRODUCTION

In recent years, glTF (gl transmission format)[1] defined by Khronos group has widely been used as a standard of 3D file format. It is easily used on a web browser and standard tools in Windows, and so on. As shown in Figure 1, glTF basically consists of a JSON-formatted file (.gltf) and binary files (.bin). A JSON-formatted file contains node hierarchy, materials, cameras, as well as descriptor information for meshes, animations, and other constructs. Binary files contain geometry and animation data, and other buffer-based data. glTF supports multiple animations. However, it is not possible to do animation based on physical parameters which is related to dynamics and environments since the animation is attached to fixed in binary files.

On the other hand, in robot developments, the 3D file format URDF (unified robot description format)[2] is used to represent robot models and is used for simulations and animations based on physics. However, the use of URDF requires an environment for robot development called ROS (robot operating systems), and it is difficult to use for non-professional-people because the supported tools also require specialic technical knowledge on ROS.

In this study, we propose a new 3D file format RDTF (robot description transmission format), which is compatible with glTF and can be used for robots simulation.

RDTF is an extension of glTF. Therefore, 3D models can be used with tools that support glTF. It is worth to mention that there are many existing glTF models[3] which can be transformed to RDTF.

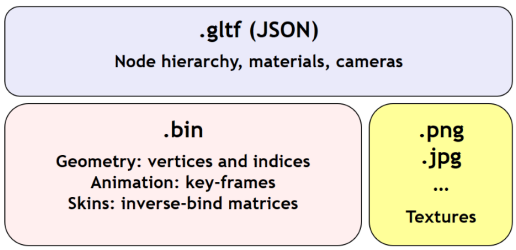


Fig. 1 glTF configuration

## 2. GLTF AND URDF

### 2.1 glTF

glTF is a JSON-formatted file format for 3D model defined by Khronos group[1] and provides version 1.0 and 2.0. RDTF is based on glTF2.0.

glTF has elements shown in List 1, and each element is counted from 0. The *children* in a *node* indicate the child nodes of the node, and glTF uses a tree structure to represent the structure of the model. Each node can define coordinate by using *translation*, *rotation* and *scale*. The *meshes* represent visual information of the model. In this example, node 1 refers to mesh 0, and mesh 0 represents visual information of node 1.

List 1 Example of glTF

```
1  {
2    "asset": { ... },
3    "scene": 0,
4    "scenes": [ ... ],
5    "nodes": [
6      {
7        "children":[1,2]
8      },
9      {
10       "mesh":0,
11       "translation":[0.0,0.0,0.0]
12     },
13     {
14       "children":[3]
15     },
16     {
17       "mesh":1,
18       "scale":[1.0,1.0,1.0]
19     }
20   ],
21   "meshes" : [ ... ],
22   "animations":[ ... ]
23       :
24   "buffers": [ ... ]
25 }
```

### 2.2 URDF

URDF is a XML-formatted file format for robot model which contains physical properties, robot appearance, part configuration, joint axis configuration, degree of freedom, and etc[2]. URDF is commonly used in ROS tools such as Rviz[6] and Gazebo[7]. The model consists of links and joints. Links describe visual information, inertia properties, and etc. Joints describe parent link, child

link, attenuation coefficients and etc.

As for the visual information in URDF, links may have multiple visuals, which contain basic primitive such as box or cylinder, or also reference of external files such as STL and COLLADA.

In the example shown in List 2, the joint between the two links is joint 1.

List 2 Example of URDF

```
<robot name="sample">
<link name="link1">
  <visual>
    <geometry>
      <box size="0.02 0.02 0.1"/>
    </geometry>
  </visual>
  <inertial>
    <origin xyz="x y z"/>
    <mass value="0.16">
    <inertia ixx="0.1" .../>
  </inertial>
</link>
<link name="link2"> ... </link>
<joint name="joint1" type="revolute">
  <parent link="link1"/>
  <child link " =link2"/>
  <dynamics damping="0.0"/>
</joint>
</robot>
```

## 3. RDTF

### 3.1 extras of glTF

glTF allows extension of the specification by using *extras* elements without any affects to many tools that support glTF. List 3 shows an outline of the extension of glTF to RDTF. Just adding extras for RDTF to List 1 makes List 3.

List 3 extras of glTF

```
{
  "asset":{ ... },
    :
  "buffers":[ ... ],
  "extras":{
    "RDTF":{ ... }
  }
}
```

### 3.2 Parameters for RDTF

We attach extra elements to glTF *nodes* by using a *node number* (integer) which indicates the target node of glTF. List 4 shows an example in which *extras* of the model.

List 4 Example of RDTF

```
"extras":{
  "RDTF":{
    "nodes":[
      {
        "node":0,
        "type":"link"
      },
      {
        "node":1,
        "type":"mesh"
      },
        :
      {
        "node":5,
        "type":"joint",
        "joint":{
          "type":"fixed",
          "friction":0.01,
          "damping":0.4
        }
      },
      {
        "node":6,
        "type":"inertial",
        "inertial":{
          "mass":1.0,
          "inertia":[0.1,0.0, ...]
        }
      },
        :
    ]
  }
}
```

Table 1 Parameters of RDTF

| | description |
|---|---|
| node | Index of the target node. |
| type | Type of the node. |
| joint | Joint properties. There are ”type(type of joint)”, ”friction”, ”damping”, and ”springStiffness”. |
| inertial | Inertial properties. There are ”mass”, ”gravityCenter”, and inertia(moment of inertia)”. |

Table 1 shows parameters of RDTF.

Table 2 Type of node

| type | description |
|---|---|
| link | Node for link. |
| joint | Node for joint. |
| mesh | Node with mesh. |
| visual | Node that manages mesh nodes. |
| inertial | Node with inertial. |

We define *type* as an extension of glTF which indicating type of node. There are ”link”, ”joint”, ”visual”, ”mesh”, and ”inertial” types as shown in Table 2.

## 4. CONVERSION FROM GLTF TO RDTF

In order to convert glTF to RDTF, we analyze and complement the nodes of glTF.

### 4.1 Analysis of nodes

We analyze the node hierarchy of glTF, and decide an appropriate *type* for each node.

Algorithm 1 shows the overall process in which Algorithm 2 and Algorithm 3 are used for the analysis of the nodes.

**Algorithm 1** Calling Algorithm2 and Algorithm3

```
INPUT=glTF.nodes
root_nodes=glTF.scenes[glTF.scene].nodes
for each node of root_nodes do
  Algorithm2(null, node)
  Algorithm3(null, node)
end for
OUTPUT=typed nodes
```

**Algorithm 2** Decision algorithm for type of mesh and visual

```
INPUT=(parent, node)
if node.mesh≠null then
  node.type ← ”mesh”
else if each child of node.children has no node.mesh ∧
each child of parent.children has no node.mesh then
  node.type ← ”visual”
end if
if node.children≠null then
  for each child of node.children do
    Algorithm2(node, child)
  end for
end if
```

**Algorithm 3** Decision algorithm for type of joint and link

```
INPUT=(parent, node)
if node.type=null then
  if size of node.children=1 ∧
  the child has no node.mesh then
    node.type ← ”joint”
  else
    node.type ← ”link”
  end if
end if
if node.children≠null then
  for each child of node.children do
    Algorithm3(node, child )
  end for
end if
```

### 4.2 Complement of nodes

We complement the nodes required for the structure of the robot. There are three complementing nodes: visual nodes, joint nodes, and inertial nodes. We apply Algorithm 4 to each node whose *type* is ”link”. When we apply the algorithm to the model shown in List 5, the nodes

- Node4(visual node of node0)
- Node5(joint node between node0 and node2)
- Node6(inertial node of node0)
- Node7(visual node of node2)
- Node8(inertial node of node2)

are complemented as shown in List.

List 5 Node complemention exsample

```
"nodes": [
  {
    "children":[4,5,6]
  },
  {
    "mesh":0,
    "translation":[0.0,0.0,0.0]
  },
  {
    "children":[3]
  },
  {
    "mesh":1,
    "scale":[1.0,1.0,1.0]
  },
  {
    "name":"node0_visual",
    "children":[1]
  },
  {
    "name":"node0_node2_joint",
    "children":[2]
  },
  {
    "name":"node0_inertial"
  },
  {
    "name":"node2_visual",
    "children":[3]
  },
  {
    "name":"node2_inertial"
  }
]
```

**Algorithm 4** node completion

```
INPUT=glTF.nodes
for each node of nodes do
  if node.type=”link” then
    for each child of node.children do
      if child.type=”mesh” then
        add child to mChildren
      else if child.type=”link” then
        add child to lChildren
      else
        add child to nChildren
      end if
    end for
    if mChildren≠null then
      visual_node.children←mChildren
      visual_node.type←”visual”
      insert(visual_node)
      add visual_node to nChildren
    end if
    if lChildren≠null then
      joint_node.children←lChildren
      joint_node.type←”joint”
      insert(joint_node)
      add joint_node to nChildren
    end if
    inertial_node.type←”inertial”
    insert(inertial_node)
    add inertial_node to nChildren
    node.children←nChildren
  end if
end for
OUTPUT=inserted nodes
```

### 4.3 Inertial properties

In this section, we propose a method for calculate the mass and centroid of the link. As shown in Figure 2, the link is composed of multiple meshes, the mesh

is composed of basic primitives such as cylinders and rectangular parallelepipeds, and the primitive is composed of facets i.e., multiple triangular polygons. Let $\ell$ be the number of meshes that make up the link, $m_i$ be the number of primitives that make up the mesh$(i)(i = 1 \dots \ell)$, and $n_j$ be the number of facets that makes up the primitive$(i, j)(j = 1 \dots m_i)$. Then considering the triangular pyramid formed by the origin and one facet, the 3D model can be divided into multiple triangular pyramids, and volume and centroid of whole model can be obtained by summing the volume and centroid of triangular pyramids.

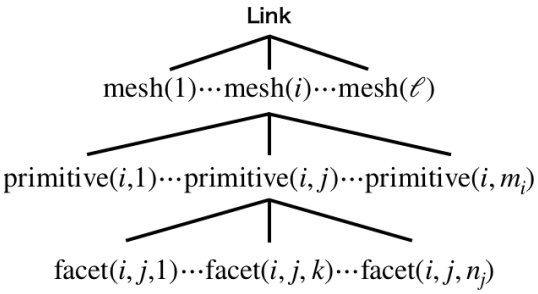


Fig. 2 Link configuration

We use *scalar triple product* of

$$\boldsymbol{A} \cdot (\boldsymbol{B} \times \boldsymbol{C}) = |\boldsymbol{A}||\boldsymbol{B} \times \boldsymbol{C}| \cos\theta \tag{1}$$

to calculate the signed volume of the parallelepiped and the 6 times triangular pyramid which are formed by the three 3-dimensional vectors $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$. At this time, $\theta$ is the angle between $\boldsymbol{A}$ and $\boldsymbol{B} \times \boldsymbol{C}$.

Next, we calculate the inertial properties for the link shown in the Figure 2. First, consider facet$(i, j, k)(k = 1 \dots n_j)$ to calculate the volume and centroid of the triangular pyramid formed by the origin and each facet. In equation(1), the facets(back) turn to outside from the origin have positive values and the facets(front) turn to the inside from the origin have negative values, and the sum of the front and back volumes is the volume of each primitive. If we define that the coordinate groups that make up facet$(i, j, k)$ are created from $\boldsymbol{a}_{ijk}, \boldsymbol{b}_{ijk}, \boldsymbol{c}_{ijk}$, and the origin define $\boldsymbol{o} = (0.0, 0.0, 0.0)$, the triangular pyramid volume $V_f(i, j, k)$ and the centroid $\boldsymbol{g}_f(i, j, k)$ are given by

$$\begin{aligned} V_f(i,j,k) &= \frac{\boldsymbol{a}_{ijk} \cdot (\boldsymbol{b}_{ijk} \times \boldsymbol{c}_{ijk})}{6} \\ \boldsymbol{g}_f(i,j,k) &= \frac{\boldsymbol{o} + \boldsymbol{a}_{ijk} + \boldsymbol{b}_{ijk} + \boldsymbol{c}_{ijk}}{4} \end{aligned}$$

Therefore, the primitive volume $V_p(i, j)$ and the centroid $\boldsymbol{g}_p(i, j)$ are given by

$$\begin{aligned} V_p(i,j) &= \sum_{k=1}^{n_j} V_f(i,j,k) \\ \boldsymbol{g}_p(i,j) &= \frac{\sum_{k=1}^{n_j} \boldsymbol{g}_f(i,j,k) V_f(i,j,k)}{V_p(i,j)} \end{aligned}$$

Next, we calculate the mesh volume and centroid. Since meshes are referred known nodes of type "mesh", we need to consider the translation and scale of the node. If we define that the $i$th mesh's translation as $\boldsymbol{t}_m(i)$, scale as $\boldsymbol{s}_m(i) = [s_m^x(i), s_m^y(i), s_m^z(i)]^{\mathrm{T}}$, and the density as $d(i)$, the $i$th mesh's mass $m(i)$ and centroid $\boldsymbol{g}_m(i)$ are as shown below. However, the operator $\circ$ shows multiplication of each element of a vector.

$$\begin{aligned} m(i) &= s_m^x(i) s_m^y(i) s_m^z(i) d(i) \sum_{j=1}^{m_i} V_p(i,j) \\ \boldsymbol{g}_m(i) &= \frac{\sum_{j=1}^{m_i} \boldsymbol{g}_p(i,j) V_p(i,j)}{\sum_{j=1}^{m_i} V_p(i,j)} \circ \boldsymbol{s}_m(i) + \boldsymbol{t}_m(i) \end{aligned}$$

Let's define the node's translation as $\boldsymbol{t_l}$ and scale as $\boldsymbol{s_l} = [s_l^x, s_l^y, s_l^z]^{\mathrm{T}}$, the link's mass $M_l$ and centroid $\boldsymbol{g_l}$ are obtained by

$$\begin{aligned} M_l &= s_l^x s_l^y s_l^z \sum_{i=1}^{\ell} m(i) \\ \boldsymbol{g_l} &= \frac{\sum_{i=1}^{\ell} \boldsymbol{g}_m(i) m(i)}{\sum_{i=1}^{\ell} m(i)} \circ \boldsymbol{s_l} + \boldsymbol{t_l} \end{aligned}$$

## 5. EXAMPLE

In this section, we show an example of the proposed 3D file format RDTF.

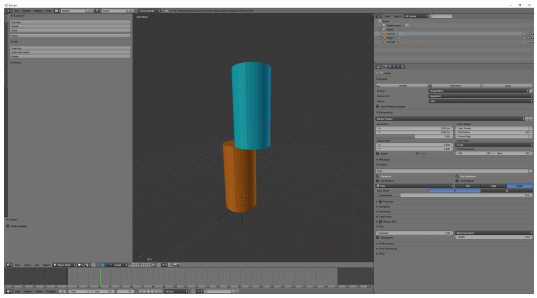

Fig. 3 two_cylinder.gltf(Blender)

### 5.1 Convert glTF to RDTF

We create a target model with Blender as shown in Figure 3, and export to glTF. Then we convert the glTF to RDTF using the proposed method. Since the RDTF use *extras* of glTF, the generated RDTF can be used as glTF. Figure 4 shows the screen shot of glTF-Viewer[8].

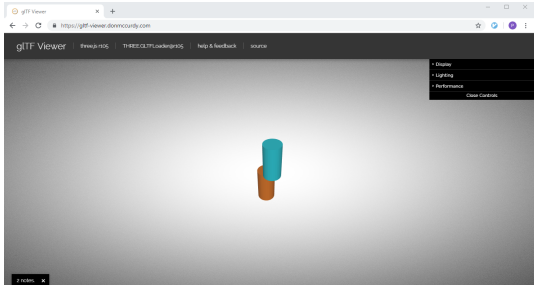


Fig. 4 two_cylinder.gltf(glTF-Viewer)

### 5.2 Convert RDTF to URDF

We convert RDTF to URDF with Mikity3D[4][5], witch generates COLLADA files for primitives.

By making the world file and launch file for Gazebo[7], we can easily perform physical simulation in Gazebo as shown in Figure 5.

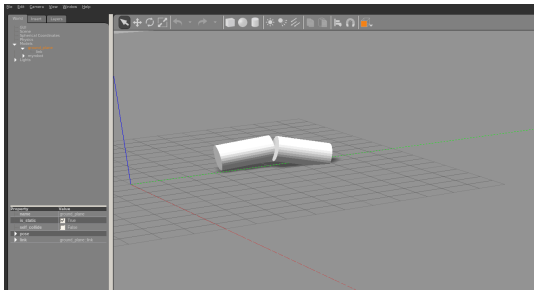

Fig. 5 Rendering with Gazebo

## 6. CONCLUSION

Since RDTF proposed in this study is an extension of glTF, we are able to use the models with many tools that supported glTF. RDTF can be converted to URDF and can be used many tool in ROS. Further studies are needed in order to deal with glTF that contains Bone and Skin, which are commonly used in 3D animation.

## REFERENCES


[1] KhronosGroup. gltf/specification/2.0. https://github.com/KhronosGroup/glTF/tree/master/specification/2.0.

[2] OpenRobotics. URDF. http://wiki.ros.org/urdf/XML/.

[3] Sketchfab. https://sketchfab.com/.

[4] Mikity3D. https://mikity3d.mk.ces.kyutech.ac.jp.

[5] Tatsuro Yunoki,Masanobu Koga,Akira Tsutsumi. Cooperation between Control System Simulation Tool Jamox and 3D Animation Tool Mikity3D Compatible with ROS Model Data. Proceedings of ... Annual Conference of the Institute of Systems, Control and Information Engineers61,4p,2017-05-23.

[6] Rviz. http://wiki.ros.org/rviz.

[7] Gazebo. http://gazebosim.org/.

[8] glTF Viewer. https://gltf-viewer.donmccurdy.com.