(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
19 December 2024 (19.12.2024)

WIPO | PCT

(10) International Publication Number
WO 2024/256340 A1

(51) International Patent Classification:
G06T 15/04 (2011.01) *H04N 19/597* (2014.01)
G06T 9/00 (2006.01)

(21) International Application Number:
PCT/EP2024/065958

(22) International Filing Date:
10 June 2024 (10.06.2024)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
23305963.3 16 June 2023 (16.06.2023) EP

(71) Applicant: **INTERDIGITAL CE PATENT HOLDINGS, SAS** [FR/FR]; 3 rue du Colonel Moll, 75017 PARIS (FR).

(72) Inventors: **GOSSELIN, Philippe Henri**; c/o InterDigital, Immeuble ZEN 2, 845A Avenue des Champs Blancs, 35510 CESSON-SEVIGNE (FR). **COVA REGATEIRO, João Pedro**; c/o InterDigital, Immeuble ZEN 2, 845A Avenue des Champs Blancs, 35510 CESSON-SEVIGNE (FR). **LE CLERC, Francois**; c/o InterDigital, Immeuble ZEN 2, 845A Avenue des Champs Blancs, 35510 CESSON-SEVIGNE (FR).

(74) Agent: **INTERDIGITAL**; Immeuble Zen 2, 845A Avenue des Champs Blancs, 35510 CESSON-SEVIGNE (FR).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MU, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, CV, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SC, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ,

(54) Title: TEXTURE TARGETS IN SCENE DESCRIPTION

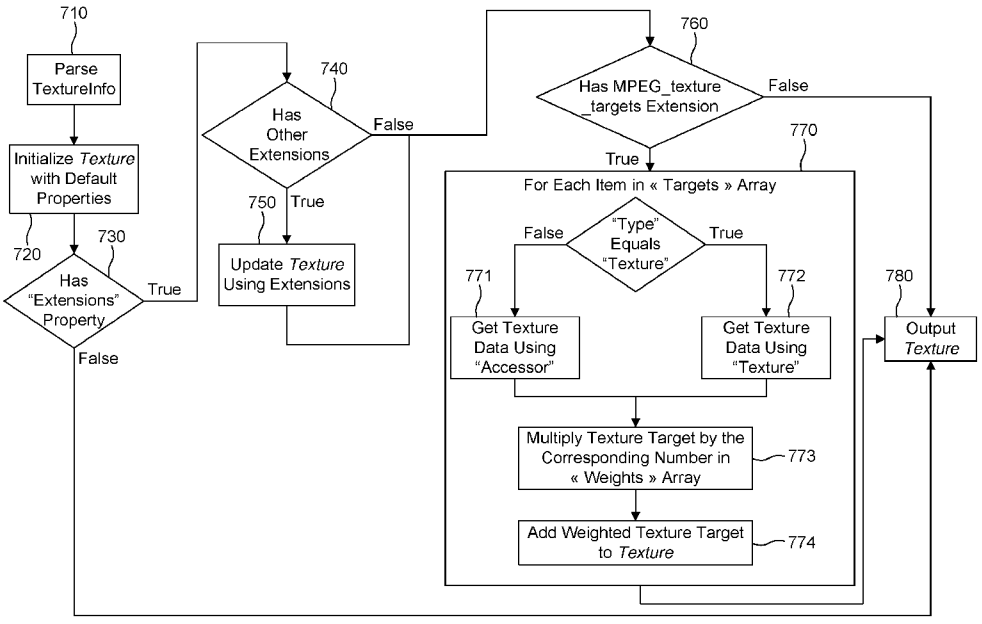


FIG. 7

(57) Abstract: In one implementation, we propose to provide signaling to inform applications about a representation of a texture as a linear combination of textures. The representation of the proposed format is reconstructed given at least one texture target with an associated target weight. This representation facilitates the transfer of parametric meshes with parametric textures.

[Continued on next page]

WO 2024/256340 A1

RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, ME, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

TEXTURE TARGETS IN SCENE DESCRIPTION

CROSS REFERENCE TO RELATED APPLICATIONS

[1] This application claims the benefit of European Patent Application No 23305963.3, filed on June 16, 2023, which is incorporated herein by reference in its entirety.

TECHNICAL FIELD

[2] The present embodiments generally relate to texture encoding/decoding for 3D scene representations, more particularly, to a format used to represent a texture as a linear combination of texture images in MPEG-I Scene Description.

BACKGROUND

[3] Three-dimensional scenes and models may be rendered in real-time to a user. Some formats such as the one proposed by Khronos / glTF (Graphics Language Transmission Format) and its extensions defined in MPEG Scene Description format or Apple / USDZ are possible ways to represent a 3D content to be rendered. Such formats support 3D model geometry, appearance, scene graph hierarchy, and animation. They are intended to be a streamlined, interoperable format for the delivery of 3D assets, while minimizing file size and runtime processing by applications. It is desirable to improve the representation of texture in such environment to offer more flexibility.

SUMMARY

[4] The drawbacks and disadvantages of the prior art are solved and addressed by the general aspects described herein.

[5] In one implementation, we propose to provide encoding/decoding of parametric texture for 3D scene representations. The new proposed format allows to encode a texture as a linear combination of texture images in MPEG-I Scene Description. We introduce the term “texture target” to designate a texture component in the combination. The new proposed format called “MPEG_texture_targets” allows the transfer of meshes with parametric textures. Besides, the proposed format is fully compatible with existing texture encoding.

[6] According to a first aspect, there is provided a method. The method comprises obtaining at least one format, from a description for a 3D scene, used to represent a texture as a linear combination of textures; obtaining, from the description, at least one texture target with an associated target weight; and reconstructing a texture based on the at least one format and the at least one texture target with its associated target weight.

[7] According to a second aspect, there is provided a method. The method comprises obtaining at least one format, from a description for a 3D scene, used to represent a texture as a linear combination of textures; obtaining, from the description, at least one texture target with an associated target weight; and encoding a texture based on the at least one format and

the at least one texture target with its associated target weight.

[8] One or more embodiments also provide a computer program comprising instructions which when executed by one or more processors cause the one or more processors to perform the method according to any of the embodiments described herein. One or more of the present embodiments also provide a computer readable storage medium having stored thereon instructions for processing scene description according to the methods described herein.

[9] One or more embodiments also provide a computer readable storage medium having stored thereon a scene description generated according to the methods described above. One or more embodiments also provide a method and apparatus for transmitting or receiving the scene description generated according to the methods described herein.

BRIEF DESCRIPTION OF THE DRAWINGS

[10] FIG. 1 shows an example architecture of 3D content creation tools and modern graphics processing engine.

[11] FIG. 2 shows an example of the syntax of a data stream encoding an 3D scene description.

[12] FIG. 3 shows an example graph of 3D scene description.

[13] FIG. 4 illustrates the relations between top-level arrays either in a glTF asset or in a MPEG-I Scene Description (SD) into which the present principles may be implemented.

[14] FIG. 5a, 5b and 5c illustrate various steps of a same animation of textures according to an embodiment.

[15] FIG. 6 illustrates a texture image used in the above animation of textures according to an embodiment.

[16] FIG. 7 illustrates an exemplary embodiment of a method for parsing a TextureInfo.

[17] FIG. 8 illustrates various textures for parametric meshes to which the processing of a texture image according to an embodiment may apply.

DETAILED DESCRIPTION

[18] **FIG. 1** shows an example architecture of 3D processing engine 130 which may be configured to implement the methods described herein. A device according to the architecture of FIG. 1 is linked with other devices via their bus 131 and/or via I/O interface 136.

[19] Device 130 comprises following elements that are linked together by a data and address bus 131:

- a microprocessor 132 (or CPU), which is, for example, a DSP (or Digital Signal Processor);
- a ROM (or Read Only Memory) 133;
- a RAM (or Random Access Memory) 134;
- a storage interface 135;

- an I/O interface 136 for reception of data to transmit, from an application; and
- a power supply (not represented in FIG. 1), e.g., a battery.

[20] In accordance with an example, the power supply is external to the device. In each of mentioned memory, the word "register" used in the specification may correspond to area of small capacity (some bits) or to very large area (e.g., a whole program or large amount of received or decoded data). The ROM 133 comprises at least a program and parameters. The ROM 133 may store algorithms and instructions to perform techniques in accordance with present principles. When switched on, the CPU 132 uploads the program in the RAM and executes the corresponding instructions.

[21] The RAM 134 comprises, in a register, the program executed by the CPU 132 and uploaded after switch-on of the device 130, input data in a register, intermediate data in different states of the method in a register, and other variables used for the execution of the method in a register.

[22] Device 130 is linked, for example via bus 131 to a set of sensors 137 and to a set of rendering devices 138. Sensors 137 may be, for example, cameras, microphones, temperature sensors, Inertial Measurement Units, GPS, hygrometry sensors, IR or UV light sensors or wind sensors. Rendering devices 138 may be, for example, displays, speakers, vibrators, heat, fan, etc.

[23] In accordance with examples, the device 130 is configured to implement a method according to the present principles, and belongs to a set comprising:

- a mobile device;
- a communication device;
- a game device;
- a tablet (or tablet computer);
- a laptop;
- a still picture camera;
- a video camera.

[24] In 3D applications, scene descriptions are used to combine explicit and easy-to-parse description of a scene structure and some binary representations of media content. **FIG. 2** shows an example of the syntax of a data stream encoding a 3D scene description. FIG. 2 also shows an example structure 210 of 3D scene description. The structure consists in a container which organizes the stream in independent elements of syntax. The structure may comprise a header part 220 which is a set of data common to every syntax element of the stream. For example, the header part comprises some of metadata about syntax elements, describing the nature and the role of each of them. The structure also comprises a payload comprising an element of syntax 230 and an element of syntax 240. Syntax element 230

comprises data representative of the media content items described in the nodes of the scene graph related to scene elements. Images, meshes and other raw data may have been compressed according to a compression method. Alternatively, images, meshes and other raw data may have been directly embedded in scene description data 240 without being compressed. Element of syntax 240 is a part of the payload of the data stream and comprises data encoding the scene description as described according to the present principles.

[25] **FIG. 3** shows an example graph 310 of a 3D scene description. In this example, the scene graph may comprise a description of real objects, for example 'plane horizontal surface' (that can be a table or a road) and a description of 3D objects 312, for example an animation of a car. Scene description is organized as an array 310 of nodes. A node can be linked to child nodes to form a scene structure 311. A node can carry a description of a real object (e.g., a semantic description) or a description of a 3D object. In the example of FIG. 3, node 301 describes a camera located in the 3D volume of the scene. Node 302 describes a car and comprises an index of a representation of the car, for example an index in an array of 3D meshes. Node 303 is a child of node 302 and comprises a description of one wheel of the car. The same way, it comprises an index to the 3D mesh of the wheel. The same 3D mesh may be used for several objects in the 3D scene as the scale, location and orientation of objects are described in the scene nodes.

[26] **FIG. 4** illustrates the current solution to represent texture in MPEG-I Scene Description (SD). In particular, FIG. 4 represents a glTF file structure. The entry point is the "scene" node (430) which contains "node" node(s) (435) that can be for instance a "camera" (410) or a "mesh" (440). The MPEG-I Scene Description (SD) format allows the storage in glTF files of texture image (498) or a link (498). The representation of textures (490) uses images (498) and textures samplers (495) in the glTF file. The first one (498) defines images loaded from a file or included in the glTF file (with buffer views (450)). The second one (495) points to samplers allowing to define how to use the image data. For instance, the MPEG-I Scene Description (SD) format allows the definition of sampling and wrapping strategies. Moreover, an extension may apply at the glTF texture level, for instance KHR_texture_transforms, which specifies more complex transforms to apply on a texture image such as rotate or rescale the image. Then, materials assets (470) in the glTF file reference textures may be used to define how the texture is used in the rendering, like changing colors, normals, etc.

[27] Today, this approach is limited as it does not provide flexibility in combining different texture images, or while trying to animate the texture. Textures combination can be performed before the creation of the glTF file in a so-called "baking" processing. During these pre-processing steps, one computes the combination and only stores the resulting texture in the

glTF file. If it allows the transportation and rendering of the final scene, as we lose all the combination process, it no longer allow any further changes in the rendering of the final scene. Animation of texture can be also performed with "baking" processing. The texture image for each frame of the animation can be precomputed and then sent alongside the glTF file. However, the skilled in the art will recognize that such processing quickly results in a very large amount of image data, and there is still no way to modify these texture images.

[28] It is therefore desirable to improve the signaling/encoding of the texture assets combining different texture images or animating the texture along with time.

[29] **MPEG_Texture_Targets extension in MPEG-I Scene Description**

[30] The proposed format according to any of the embodiments disclosed hereafter is advantageously compliant with the glTF format and with the recent MPEG-I SD effort to extend glTF with MPEG-I SD extensions or with any other 3D scene description format. The skilled in the art will recognize that the meaning and use of the disclosed embodiments are generic and may be coded in other formats (XML, USD, ...).

[31] In the current MPEG-I or in the current version of glTF and available extensions, a combination of different texture images or animation of the texture has to be defined and computed in advance which is not convenient for adapting the texture to a parametric mesh for instance. Therefore, one embodiment proposes to introduce the definition of a weighted sum of texture images in the glTF file:

$$texture = clamp(textureBase + \sum_i weight_i \times textureTarget_i) \qquad (1)$$

[32] With this new proposed definition, a weighted sum of images (such as 2D arrays of vectors) is processed. According to a particular variant, the values resulting from the weighted sum are clamped to ensure that they are in a valid color range. With a range defined as [0,1], the clamping operation turns any negative value to zero and any value higher than one into one. Consequently, the proposed encoding allows the creation of new materials based on the weighted combination of texture images.

[33] The *textureBase* and *textureTarget*s in the above equation (1) are textures, accessors or images as defined in the glTF file using current standard and extensions (respectively in *textures, accessors* or *images* assets).

[34] We also assume that the set of texture bases and targets that we consider in combination, have the same layout (same width, height and vector size/channel count), whatever the processes that build them. That includes extensions like KHR_texture_transform that can change further the layout.

[35] The glTF Specification makes use of common engineering and graphics terms such as image, buffer, texture, etc. to identify and describe certain glTF constructs and their attributes, states, and behaviors. The specification defines a texture as an object that

combines an image and its sampler (as disclosed in the Section 5.29 “Texture” of the glTF 2.0 specification (https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.pdf)) that allows the definition of sampling and wrapping strategies.

Name	Type	Description	Required
sampler	integer	The index of the sampler used by this texture. When undefined, a sampler with repeat wrapping and auto filtering SHOULD be used.	No
source	integer	The index of the image used by this texture. When undefined, an extension or other mechanism SHOULD supply an alternate texture source, otherwise behavior is undefined.	No
name	string	The user-defined name of this object.	No
extensions	extension	JSON object with extension-specific objects.	No
extras	extras	Application-specific data.	No

Table 1: The GlTF Texture property

[36] The ”*source*” property of Texture references the base texture. According to a particular embodiment, it may correspond to the base texture in the equation (1).

[37] Besides, in a glTF file, one can reference a texture using a *TextureInfo* property. It is used in several places in *materials* assets, like “baseColorTexture”, “normalTexture”, or “metallicRoughnessTexture”. The specification defines four properties for TextureInfo: “index”, “texCoord”, “extensions” and “extras” as disclosed in Section 5.30 “TextureInfo” of the glTF 2.0 specification (https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.pdf) which is reproduced below:

Name	Type	Description	Required
index	integer	The index of the texture	Yes
texCoord	integer	The set index of texture’s TEXCOORD attribute used for texture coordinate mapping.	No, default: 0
extensions	extension	JSON object with extension-specific objects.	No
extras	extras	Application-specific data.	No

Table 2: The GlTF TextureInfo property

[38] The ”*index*” property of *TextureInfo* references the base texture. According to a particular embodiment, it may correspond to the base texture in the equation (1).

[39] According to one embodiment, a new “MPEG_texture_targets” property is defined in the “extensions” attribute respecting the definition of the TextureInfo property. According to another embodiment, the new “MPEG_texture_targets” property may be defined in the “extensions” attribute of the Texture property. The skilled in the art will recognize that this new

"MPEG_texture_targets" property is optional.

[40] The "MPEG_texture_targets" property contains two arrays of properties:

Name	Type	Description	Required
targets	TextureTarget[1-*]	List of texture targets (*TextureInfo*)	No
weights	number [1-*]	List of target weights (numbers)	No

Table 3: The MPEG_texture_targets property

[41] The "targets" property is an array of *TextureTarget*. It contains all the information needed to reference, and eventually update, a texture. Each item of index i in this array corresponds to $textureTarget_i$ in equation (1).

[42] The "weights" property is an array of float numbers to weight each texture target. Each item of index i in this array corresponds to $weight_i$ in equation (1).

[43] According to a particular feature, the size of the arrays "targets" and "weights" are the same.

[44] According to yet another particular feature, a *TextureTarget* property defines the texture data for one texture target.

Name	Type	Description	Required
type	string	Defines which attribute to use: "texture" or "accessor" or "image"	No
texture	TextureInfo	A texture	No
index	number	An accessor id or image id	No

Table 4: The TextureTarget property

[45] There are three possible sources:

[46] If "type" equals "texture", then the texture data is loaded using the "texture" property. It is a standard *TextureInfo* property.

[47] If "type" equals "image", then the texture data is loaded using the "*index*" property. It is a number that references one the images in the glTF file.

[48] If "type" equals "accessor", then the texture data is loaded using the "*index*" property. It is a number that references one the accessors in the glTF file.

[49] Whatever the source type, the layout of the data contained in the referenced accessor must be compatible with the base texture. For instance, if the base texture is a 512x512 RGB image, then a "texture" source must also be a 512x512 RGB image and an "accessor" source must be an array of 262144 (=512x512) 3D vectors.

[50] Advantageously, the disclosed format provides high compatibility with existing implementation as a particular software that does not support the texture targets extension, may use the base texture as defined in the standard, ignoring the targets. Consequently, an

incomplete but relevant rendering is still possible, assuming that in most texture combinations, the base texture is the main contributor, and the targets slightly change the final result. The skilled in the art will appreciate that, in the case where an item of “targets” has a property "texture" of type TextureInfo, it can define values in its “extensions” property in a recursive fashion, leading to an iterated specific processing for each texture target. For instance, each item in the “targets” array can use the KHR_texture_transform extension to transform texture targets before combination with the weighted sum.

[51] glTF schemas examples

[52] For the sake of clarity, these examples only show parts of the glTF file related to materials. We assume that there is a scene with one (or more) mesh that uses this material. Consequently, the following only describes the material loading procedure.

[53] According to a first exemplary embodiment, the following glTF may define a material with a combination of three texture images for the base color.

```
{
 "images": [
  {
   "uri": "textureBase.jpg"
  },
  {
   "uri": "textureTarget0.jpg"
  },
  {
   "uri": "textureTarget1.jpg"
  }
 ],
 "textures": [
  {
   "source": 0
  },
  {
   "source": 1
  },
  {
   "source": 2
  }
```

```
 ],
 "materials": [
  {
   "pbrMetallicRoughness":
   {
    "baseColorTexture":
    {
     "index": 0,
     "extensions":
     {
      "MPEG_texture_targets":
      {
       "targets": [
        {
         "type": "texture",
         "texture":
         {
          "index": 1
         }
        },
        {
         "type": "texture",
         "texture": {
          "index": 2
         }
        }
       ],
       "weights": [-0.1, 0.4]
      }
     }
    }
   }
  }
 ]
}
```

Example 1: a material with a combination of three texture images for the base color

[54] The material loader first parses the “images” list and load the three image files "textureBase.jpg", “textureTarget0.jpg” and “textureTarget1.jpg”. Then, it parses the “textures” list and assigns each texture to one of the images:

texture 0 corresponds to image in "textureBase.jpg" file;

texture 1 corresponds to image in "textureTarget0.jpg" file;

texture 2 corresponds to image in "textureTarget1.jpg" file.

[55] During the next stage, the material loader parses the “materials” list, and finds a single one with a "pbrMetallicRoughness" attribute. This attribute only defines the base color texture in the “baseColorTexture” attribute. This is a TextureInfo property: the “index” attribute defines the base texture to be used. Since “index” equals 0, it corresponds to texture 0, so the image in the “texture_base.jpg” file.

[56] Then, if the material loader supports the proposed “MPEG_texture_targets” extension, it parses the “MPEG_texture_targets” property in “materials/pbrMetallicRoughness/baseColorTexture/extensions”. The “targets” list contains two texture targets:

texture target 0 references texture 1 with weight -0.1;

texture target 1 references texture 2 with weight 0.4.

[57] As a result, the base color texture of material 0 is then the result of the following sum:

$$baseColorTexture = clamp(textureBase.jpg - 0.1 \times textureTarget0.jpg + 0.4 \times textureTarget1.jpg)$$

[58] The resulting texture is a usual 2D array of RGB vectors that can be processed by current glTF decoders and renderers.

[59] According to a second exemplary embodiment, the glTF may define a material creating an animation with textures. **FIG. 5a, 5b and 5c** illustrate various steps of a same animation of textures resulting from the processing of a texture image as shown on **FIG. 6** according to a particular embodiment. In this second example, one may create a material that changes with time, starting with a glowing rectangle as shown on FIG. 5a. This material may be applied on any surface, for instance on a cube or a mesh with the shape of a screen. In a second step, we want that a glowing number “2” slowly appears (e.g. the number fades from black to full white). FIG. 5b shows the animated texture with the number “2” fully visible. To that end, a number “2” texture is added to the rectangle texture of FIG. 5a. Then, we want that the number slowly disappears, fading back to black and we may want to repeat the same fading in/out with the number “1” and the message “GO!”. FIG. 5c shows the animated texture with the number "1" then "GO!" are fully visible. The animation ends with a final fading of the “GO!” message, going back to the initial state of FIG. 5a with an empty rectangle.

[60] **FIG. 6** illustrates a texture image used in the above animation of textures according to an embodiment. Advantageously, the present principles allow creating this animation with a single texture image. This single texture image is a texture atlas as it may contain four textures in a single image. In the glTF file, we therefore define what part of the image to use in each case. We may encode the material with the following glTF content:

```
{
 "images": [
  {
   "uri": "textureAtlas.jpg"
  }
 ],
 "textures": [
  {
   "source": 0
  }
 ],
 "materials": [
  {
   "emissiveTexture":
   {
    "index": 0,
    "extensions":
    {
     "KHR_texture_transform": {
      "offset": [0, 0],
      "scale": [0.5, 0.5]
     },
     "MPEG_texture_targets": {
      "targets": [
       {
        "type": "texture",
        "texture":
        {
         "index": 0,
         "extensions" : {
```

```
            "KHR_texture_transform": {
              "offset": [0, 0.5],
              "scale": [0.5, 0.5]
            }
          }
        }
      },
      {
        "type": "texture",
        "texture":
        {
          "index": 0,
          "extensions" : {
            "KHR_texture_transform": {
              "offset": [0.5, 0],
              "scale": [0.5, 0.5]
            }
          }
        }
      },
      {
        "type": "texture",
        "texture":
        {
          "index": 0,
          "extensions" : {
            "KHR_texture_transform": {
              "offset": [0.5, 0.5],
              "scale": [0.5, 0.5]
            }
          }
        }
      }
    ],
    "weights": [0, 0, 0]
  }
```

```
      }
     }
    }
  ],
  "animations": [
   {
    "channels": [
     {
      "sampler": 0,
      "target": {
       "path": "pointer",
       "extensions": {
        "KHR_animation_pointer": {
         "pointer":
"/materials/0/emissiveTexture/extensions/MPEG_texture_targets/weights"
        }
       }
      }
     }
    ],
    "samplers": [
     {
      "input": 0,
      "interpolation": "LINEAR",
      "output": 1
     }
    ]
   }
  ]
}
```

Example 2: a material with a combination of portions of a single texture image to create animated textures

[61] The glTF loaders first decodes the “images” and “textures” properties to read the texture image. As we have a single texture in this example, we have a single texture image with index 0.

[62] Then, the "materials" property is analyzed to find a single emissive texture. The "emissiveTexture" is a TextureInfo: its "index" property with value 0 points to the texture image. Its "extensions" property has two items: "KHR_texture_transform" and "MPEG_texture_targets".

[63] According to a preferred variant, other extensions, in this exemplary case, "KHR_texture_transform" are firstly decoded. The "offset" and "scale" properties define a texture transform that crops the top-left part of the texture atlas, corresponding to the empty glowing rectangle. Next the loader decodes "MPEG_texture_targets" extension. There are three targets: each one references the same texture ("index" is 0). They also all have "KHR_texture_transform" content that describe a crop. The first one crops the top-right part (number "2"), the second one the bottom-left part (number "1") and the last one the bottom-right part (message "GO!"). The "weights" property defines the amount of each texture target to add.

[64] An item in the "animations" list defines a single animation that updates the material. In this animation, there is a single channel in the "channels" list. It references the sampler defined in the "samplers" list and use the "KHR_animation_pointer" extension to point to the "weights" property of the "MPEG_texture_targets" of the emissive texture. The sampler references the temporal values (in "input") and weight values (in "output"). For sake of clarity, we don't show the accessors/buffer views/buffers with theses values but present their content here.

[65] The values pointed by "input" are:

[0, 1, 2, 3, 4, 5, 6]

[66] They correspond to the main steps of the animation (in seconds). The values pointed by "output" are:

[[0,0,0], [1,0,0], [0,0,0], [0,1,0], [0,0,0], [0,0,1], [0,0,0]].

[67] Each value in a weight combination (like $w_0, w_1, w_2 = [0,1,0]$) corresponds to one of the texture targets: the one with number "2", the one with number "1" and the one with message "GO!". The final texture is always the base texture (the one with a rectangle) to which we add the texture targets with weights:

$$texture = texture_{rectangle} + w_0 \times texture_{number2} + w_1 \times texture_{number1} + w_2 \times texture_{messageGO}$$

[68] They lead to the following changes, knowing that a linear interpolation is made between each step:

Time 0s: all weights are zero: only the base texture (with the rectangle) is used.

Time 1s: weights are [1,0,0]: the texture with number "2" is fully added to the base texture.

Time 2s: all weights are zero: only the base texture (with the rectangle) is used.

Time 3s: weights are [0,1,0]: the texture with number “1” is fully added to the base texture.

Time 4s: all weights are zero: only the base texture (with the rectangle) is used.

Time 5s: weights are [0,0,1]: the texture with number “GO!” is fully added to the base texture.

Time 6s: all weights are zero.

[69] The skilled in the art will appreciate the compacity of the data needed to generate this texture animation according to the at least one embodiment. Only 4 textures (grouped in a single atlas image) are required to create the animation thanks to the proposed extension. On the contrary, without the texture targets, and a frame rate of 60 images per seconds, we must bake a total of 60 x 6 - 3 = 357 textures.

[70] According to a third exemplary embodiment, the glTF may define a material where the texture sources are obtained from accessors.

```
{
  "images": [
    {
      "uri": "textureBase.jpg"
    }
  ],
  "textures": [
    {
      "source": 0
    }
  ],
  "buffers": [
    {
      "byteLength": 1179648,
      "uri": "textureTargets.bin"
    }
  ],
  "bufferViews": [
    {
      "buffer": 0,
      "byteLength": 589824,
```

```
    "byteOffset": 0,
    "name": "textureTarget0",
    "target": 34962
   },
   {
    "buffer": 1,
    "byteLength": 589824,
    "byteOffset": 589824,
    "name": "textureTarget1",
    "target": 34962
   }
  ],
  "accessors": [
   {
    "bufferView": 0,
    "byteOffset": 0,
    "componentType": 5126,
    "count": 65536,
    "name": "textureTarget0",
    "type": "VEC3"
   },
     {
    "bufferView": 0,
    "byteOffset": 0,
    "componentType": 5126,
    "count": 65536,
    "name": "textureTarget1",
    "type": "VEC3"
   }
  ],
  "materials": [
   {
    "pbrMetallicRoughness":
    {
     "baseColorTexture":
     {
```

```
        "index": 0,
        "extensions":
        {
         "MPEG_texture_targets":
         {
          "targets": [
           {
            "type": "accessor",
            "accessor": 0
           },
           {
            "type": "accessor",
            "accessor": 1
           }
          ],
          "weights": [-0.1, 0.4]
         }
        }
       }
     }
   }
  ]
}
```

Example 3: a material where the texture sources are obtained from accessors

[71] The glTF loaders first decodes the “images” and “textures” properties to read the texture image. We have a single texture in this example with index 0. This texture is a 256x256 RGB image.

[72] Then, it decodes the "buffers", "bufferViews" and "accessors" properties. The result is two arrays of 65536 (=256*256) 3D vectors with 32-bit float values:

Accessor 0: first array of 65536 3D vectors with 32-bit float values;

Accessor 1: second array of 65536 3D vectors with 32-bit float values.

[73] Then, if the material loader supports the proposed “MPEG_texture_targets” extension, it parses the “MPEG_texture_targets” property in “materials/pbrMetallicRoughness/baseColorTexture/extensions”. The “targets” list contains two texture targets:

texture target 0 references accessor 0 with weight -0.1;

texture target 1 references accessor 1 with weight 0.4.

[74] As a result, the base color texture of material 0 is then the result of the following sum:

$$baseColorTexture = clamp(textureBase.png - 0.1 \times accessor_0 + 0.4 \times accessor_1)$$

[75] The resulting texture is a usual 2D array of RGB vectors that can be processed by current glTF decoders and renderers. Advantageously, the embodiment based on "accessor" properties allows processing texture data with 32-bit float values. As usual textures are defined from 8-bit integer converted to float, the embodiment based on "texture" property is limited to 8-bit precision.

[76] Processing Model

[77] **FIG. 7** illustrates an exemplary embodiment of a method for parsing a TextureInfo. The proposed extension data is parsed any time a *TextureInfo* property occurs in the glTF file. Most glTF parsers already offer hook mechanisms one can register. For our case, we ask the main glTF parser to call the extension code every time it finds a *TextureInfo* property. As shown on FIG.7, the process ingests *TextureInfo* data (710) and outputs a texture (780).

[78] In a first step (720), the method first parses the data in a default fashion, as defined without any extension. This step (720) initializes the main texture. Then, the extension property of the texture is tested (730). If there is no "extensions" property, the process is over and outputs (780) the main texture. If there are extensions, it first applies any non-MPEG_texture_targets extensions (740, 750), like KHR_texture_transform. This holds for all current extensions but might not be true for future extensions. For these later cases, it will be up to the designers to choose the extension order. If no order is defined, we assume that MPEG_texture_targets extension always comes last. Each extension updates the main texture: its layout and values can change, but still remains a texture (e.g. a 2D array of vectors).

[79] If the MPEG_texture_targets extension is defined (760) in the *TextureInfo* data, the process (770) parses each item in the "targets" property. Since each item of this array is a *TextureInfo*, we repeat all the current processes presented with Fig. 7 in an iterated fashion. It means that each texture target (771, 772) may be transformed or the result of a combination in a specific manner. From our process point of view, it does not matter since this process always outputs a texture. The only assumption is that each texture target has the same layout as the main texture. Then, each texture target is multiplied (773) by the corresponding weight in the "weights" array and added (774) to the main texture. Once all targets are processed, the main texture is returned (780).

[80] These and other aspects, features and advantages of the general aspects will become apparent from the following detailed advantages of exemplary embodiments, which is to be

read in connection with the accompanying drawings.

[81] An exemplary embodiment relates to texture animation. Using the current glTF standard and extensions, texture can be animated with a pre-computation of all steps. Then, all these steps are stored in the glTF file, and the texture is changed every frame, leading to high memory footprint that may not fit on small hardware. Advantageously, when the animation is expressed or approximated as a linear combination of textures, as for instance exposed with the non-limiting example 2, the proposed extension dramatically reduces the size of scene data. With this approach, we only store the texture we need to, as well as the weights and animation data that reproduce the texture animation.

[82] Another exemplary embodiment relates to light effects baking. A common technique to render scene with complex light effects uses texture baking. It precomputes the lights received by scene objects and store them in textures. Then, the renderer no more needs to compute lights and only must apply the texture to get a high-quality result. This is a common approach in video games, that consoles to display rendering they cannot compute in real time. This technique works fine if they are no (or few) dynamism in scene lights. If they are a lot of changes, like unstable lights (campfire, candles, ...), explosions, objects collapsing, etc., then we must regularly bake a lot of textures to reflect the changes. In the worst case, textures must be baked every frame. This leads to very large glTF files, and texture data no more fits on small hardware. Advantageously, the proposed extension allows approximating the texture changes with combinations. For instance, unstable lights can be approximated with a dozen of textures, each one corresponding to a direction and hue. Then, for each frame, we choose a combination of these textures. We can repeat this approach every time the true texture may be approximated with a linear combination of texture targets. The resulting glTF file is then small and the scene can fit on small hardware.

[83] Another exemplary embodiment relates to the handling of numerous similar objects or characters. Indeed, some scenes can contain many similar objects or characters. For instance, one can find hundreds of trees in a forest or many warriors on a battlefield. Without the proposed extension, if we want to have unique objects or characters, we must precompute and store a different texture for each of them. As in the previous cases, it quickly leads to very large glTF files, and its data may not fit on small hardware. Advantageously, the proposed extension allows using a limited number of texture targets, and creating a unique combination for each object or character in the scene. With this approach, each new item only costs a few texture target weights.

[84] **FIG. 8** illustrates various textures for parametric meshes to which the processing of a texture image according to an embodiment may apply. There are many ways to create these texture combinations: we present an example here to create a unique texture for the face of

each character in a scene using the FLAME face model as disclosed in "*FLAME, Learning a model of facial shape and expression from 4D scans", by*. Tianye Li, Timo Bolkart, Michael J Black, Hao Li and Javier Romero (SIGGRAPH ASIA 2017, BANGKOK, THAILAND). The FLAME face model defines a parametric face rig. Face meshes can be created combining many morph targets. These meshes can be textured combining a base texture and 200 texture targets with 512x512 float values. A combination 810, 820 represents a particular face identity (skin color, eye browns, etc.) as illustrated on FIG. 8. One can store a morph targets combination in a glTF file using the standard MPEG-I SD features. Advantageously, the proposed extension allows storing a texture combination in the same glTF file. As a result, we can create scenes with thousands of characters with a small memory footprint.

[85] Furthermore, such glTF files may be transported in many ways (drivers, networks, etc.) and users receiving them can render faces and modify the combination if needed. For instance, artists in CGI production receive drafts of characters and refine them (e.g., change the combination weights) in their software (Blender, Maya, Unity, etc.).

[86] Furthermore, using extensions like KHR_animation_pointers, one can animate the texture combination, for instance to render a character whose face skin is changing with time (new hair, tattoos, scars, etc.).

[87] Various numeric values are used in the present application. The specific values are for example purposes and the aspects described are not limited to these specific values.

[88] Various methods are described herein, and each of the methods comprises one or more steps or actions for achieving the described method. Unless a specific order of steps or actions is required for proper operation of the method, the order and/or use of specific steps and/or actions may be modified or combined. Additionally, terms such as "first", "second", etc. may be used in various embodiments to modify an element, component, step, operation, etc., such as, for example, a "first decoding" and a "second decoding". Use of such terms does not imply an ordering to the modified operations unless specifically required. So, in this example, the first decoding need not be performed before the second decoding, and may occur, for example, before, during, or in an overlapping time period with the second decoding.

[89] The implementations and aspects described herein may be implemented in, for example, a method or a process, an apparatus, a software program, a data stream, or a signal. Even if only discussed in the context of a single form of implementation (for example, discussed only as a method), the implementation of features discussed may also be implemented in other forms (for example, an apparatus or program). An apparatus may be implemented in, for example, appropriate hardware, software, and firmware. The methods may be implemented in, for example, an apparatus, for example, a processor, which refers to processing devices in general, including, for example, a computer, a microprocessor, an

integrated circuit, or a programmable logic device. Processors also include communication devices, for example, computers, cell phones, portable/personal digital assistants ("PDAs"), and other devices that facilitate communication of information between end-users.

[90] Reference to "one embodiment" or "an embodiment" or "one implementation" or "an implementation", as well as other variations thereof, means that a particular feature, structure, characteristic, and so forth described in connection with the embodiment is included in at least one embodiment. Thus, the appearances of the phrase "in one embodiment" or "in an embodiment" or "in one implementation" or "in an implementation", as well any other variations, appearing in various places throughout this application are not necessarily all referring to the same embodiment.

[91] Additionally, this application may refer to "determining" various pieces of information. Determining the information may include one or more of, for example, estimating the information, calculating the information, predicting the information, or retrieving the information from memory.

[92] Further, this application may refer to "accessing" various pieces of information. Accessing the information may include one or more of, for example, receiving the information, retrieving the information (for example, from memory), storing the information, moving the information, copying the information, calculating the information, determining the information, predicting the information, or estimating the information.

[93] Additionally, this application may refer to "receiving" various pieces of information. Receiving is, as with "accessing", intended to be a broad term. Receiving the information may include one or more of, for example, accessing the information, or retrieving the information (for example, from memory). Further, "receiving" is typically involved, in one way or another, during operations, for example, storing the information, processing the information, transmitting the information, moving the information, copying the information, erasing the information, calculating the information, determining the information, predicting the information, or estimating the information.

[94] It is to be appreciated that the use of any of the following "/", "and/or", and "at least one of", for example, in the cases of "A/B", "A and/or B" and "at least one of A and B", is intended to encompass the selection of the first listed option (A) only, or the selection of the second listed option (B) only, or the selection of both options (A and B). As a further example, in the cases of "A, B, and/or C" and "at least one of A, B, and C", such phrasing is intended to encompass the selection of the first listed option (A) only, or the selection of the second listed option (B) only, or the selection of the third listed option (C) only, or the selection of the first and the second listed options (A and B) only, or the selection of the first and third listed options (A and C) only, or the selection of the second and third listed options (B and C) only, or the

selection of all three options (A and B and C). This may be extended, as is clear to one of ordinary skill in this and related arts, for as many items as are listed.

[95] As will be evident to one of ordinary skill in the art, implementations may produce a variety of signals formatted to carry information that may be, for example, stored or transmitted. The information may include, for example, instructions for performing a method, or data produced by one of the described implementations. For example, a signal may be formatted to carry the bitstream of a described embodiment. Such a signal may be formatted, for example, as an electromagnetic wave (for example, using a radio frequency portion of spectrum) or as a baseband signal. The formatting may include, for example, encoding a data stream and modulating a carrier with the encoded data stream. The information that the signal carries may be, for example, analog or digital information. The signal may be transmitted over a variety of different wired or wireless links, as is known. The signal may be stored on a processor-readable medium.

CLAIMS

1. A method, comprising:

obtaining at least one format, from a description for a 3D scene, used to represent a texture as a linear combination of textures;

obtaining, from said description, at least one texture target with an associated target weight; and

reconstructing said texture based on said at least one format and said at least one texture target with said associated target weight.

2. A method, comprising:

indicating at least one format, in a description for an extended reality scene, used to represent a texture as a linear combination of textures;

indicating at least one texture target with an associated target weight; and

encoding said texture based on said at least one format and said at least one texture target with said associated target weight.

3. The method of any one of claims 1-2, wherein said at least one format includes an array of texture targets.

4. The method of any one of claims 1-3, wherein said at least one format includes an array of target weights.

5. The method of any one of claims 1-4, further comprising

obtaining a base texture;

obtaining a linear combination of textures as a weighted sum of said at least one texture target multiplied by the associated target weight with said base texture.

6. The method of claim 5, further comprising clamping the weighted sum to obtain a linear combination of textures.

7. The method of any one of claims 1-6, wherein a texture target is obtained from the at least one format, in a second description for a 3D scene, used to represent a texture as a linear combination of texture targets.

8. An apparatus, comprising one or more processors and at least one memory coupled to said one or more processors, wherein said one or more processors are configured to:

obtain at least one format, from a description for a 3D scene, used to represent a texture as a linear combination of textures;

obtain, from said description, at least one texture target with an associated target weight; and

reconstruct said texture based on said at least one format and said at least one texture target with said associated target weight.

9. An apparatus, comprising one or more processors and at least one memory coupled to said one or more processors, wherein said one or more processors are configured to:

indicate at least one format, in a description for an extended reality scene, used to represent a texture as a linear combination of textures;

indicate at least one texture target with an associated target weight; and

encode said texture based on said at least one format and said at least one texture target with said associated target weight.

10. The apparatus of any one of claims 8-9, wherein said at least one format includes an array of texture targets.

11. The apparatus of any one of claims 8-9, wherein said at least one format includes an array of target weights.

12. The apparatus of any one of claims 8-9, wherein said one or more processors are configured to:

obtain a base texture;

obtain a linear combination of textures as a weighted sum of said at least one texture target multiplied by the associated target weight with said base texture.

13. The apparatus of claim 12, wherein said one or more processors are configured to clamp the weighted sum to obtain a linear combination of textures.

14. The apparatus of any one of claims 8-13, wherein a texture target is obtained from the at least one format, in a second description for a 3D scene, used to represent a texture as a linear combination of texture targets.

15. A non-transitory computer readable medium comprising instructions which, when the instructions are executed by a computer, cause the computer to perform the method of any of claims 1-7.

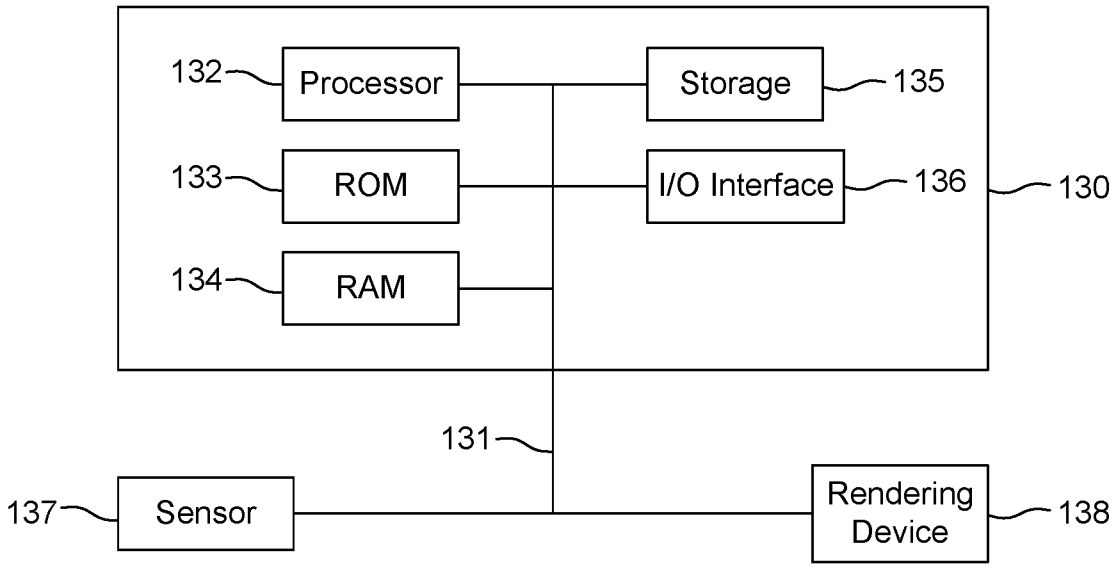

132
Processor
Storage
135
133
ROM
I/O Interface
136
130
134
RAM
131
137
Sensor
Rendering Device
138

FIG. 1

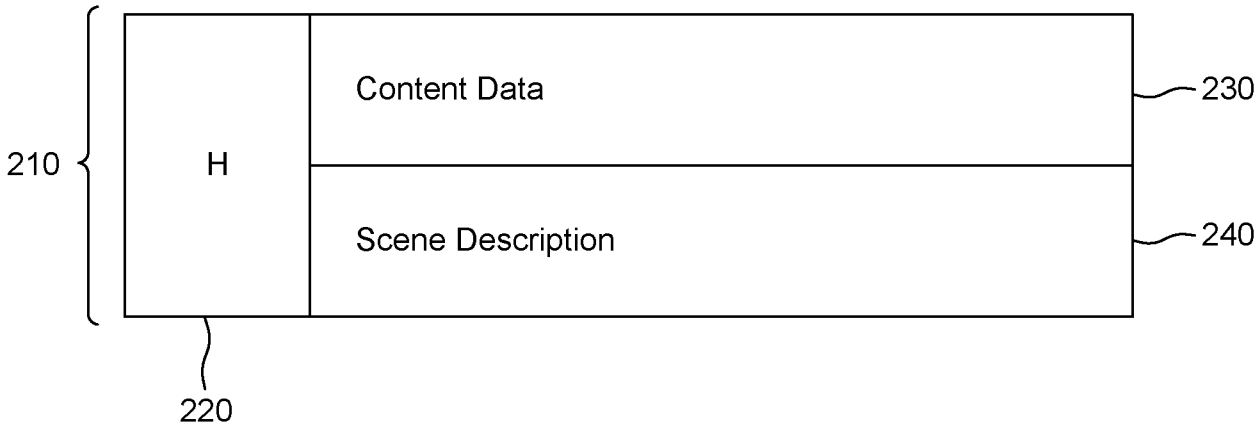

210
H
Content Data
230
Scene Description
240
220

FIG. 2

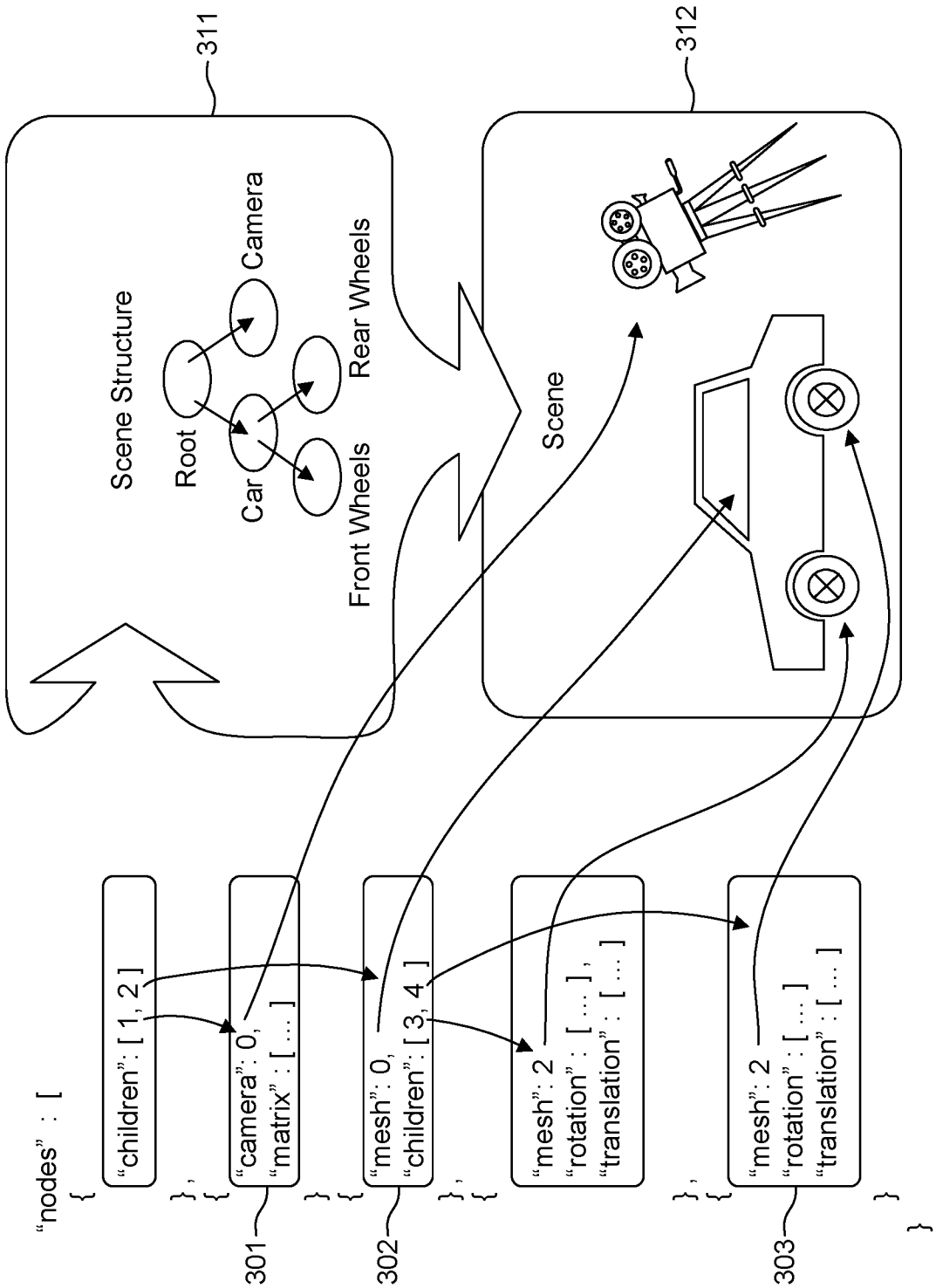

311
312
Scene Structure
Root
Camera
Car
Front Wheels
Rear Wheels
Scene
"nodes" : [
{
"children": [1, 2]
},
{
"camera": 0,
"matrix" : [...]
},
{
"mesh": 0,
"children": [3, 4]
},
{
"mesh": 2
"rotation" : [...],
"translation" : [...]
},
{
"mesh": 2
"rotation" : [...]
"translation" : [...]
}
}
301
302
303

FIG. 3

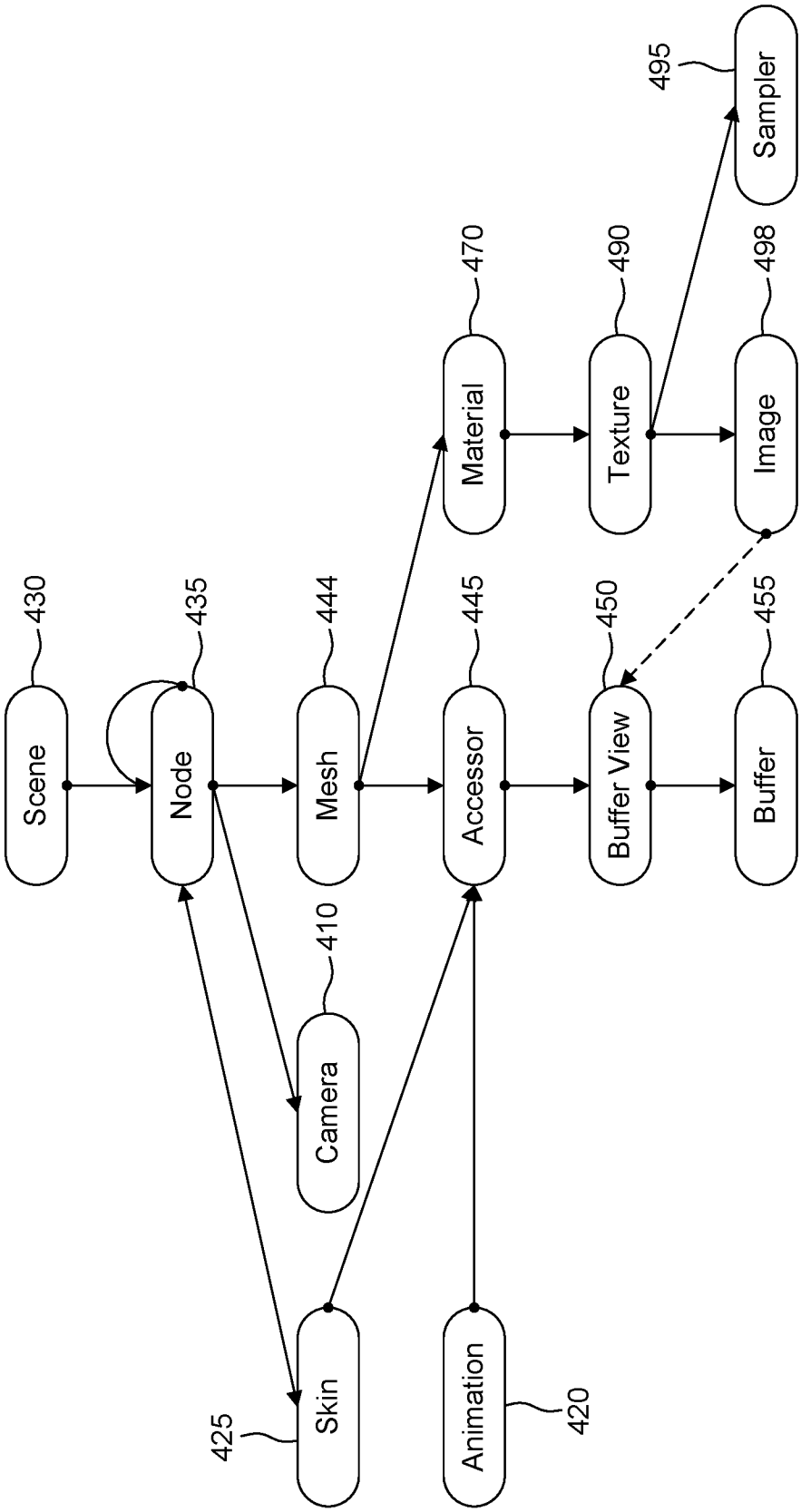

Scene
430
Node
435
Mesh
444
Material
470
Texture
490
Sampler
495
Image
498
Accessor
445
Buffer View
450
Buffer
455
Camera
410
Skin
425
Animation
420

FIG. 4

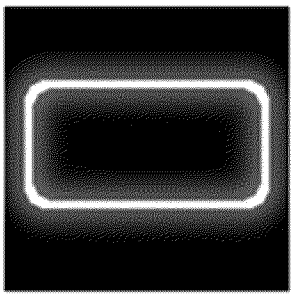


FIG. 5A

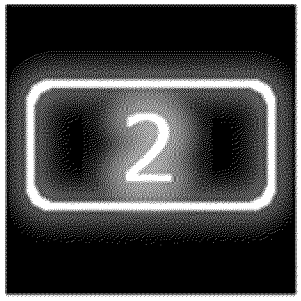
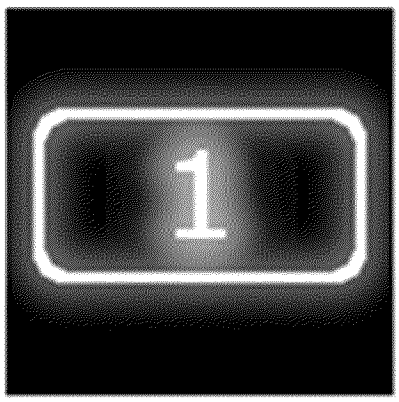


FIG. 5B



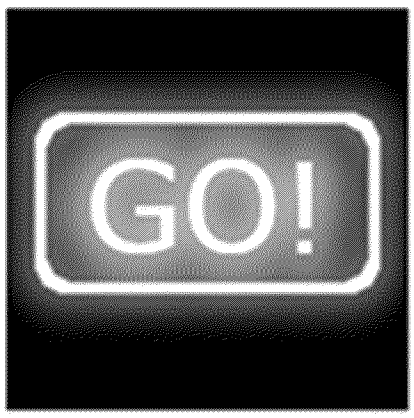


FIG. 5C

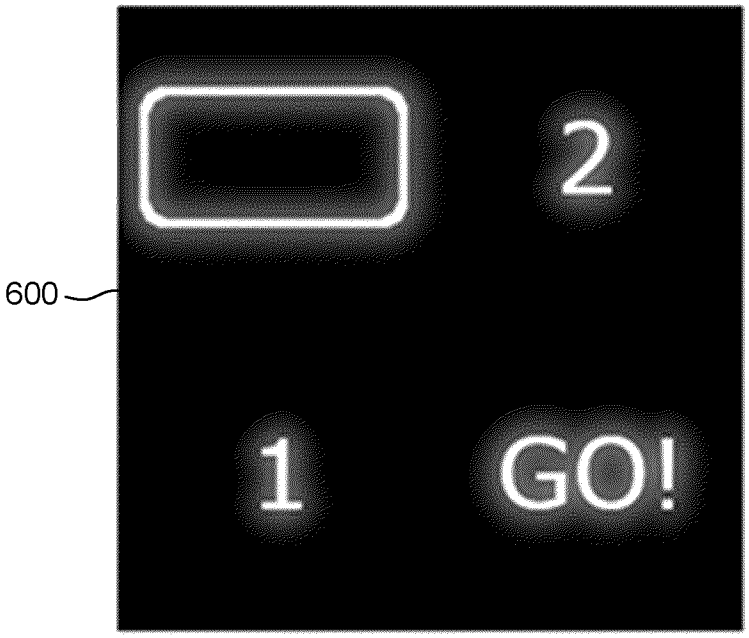


FIG. 6

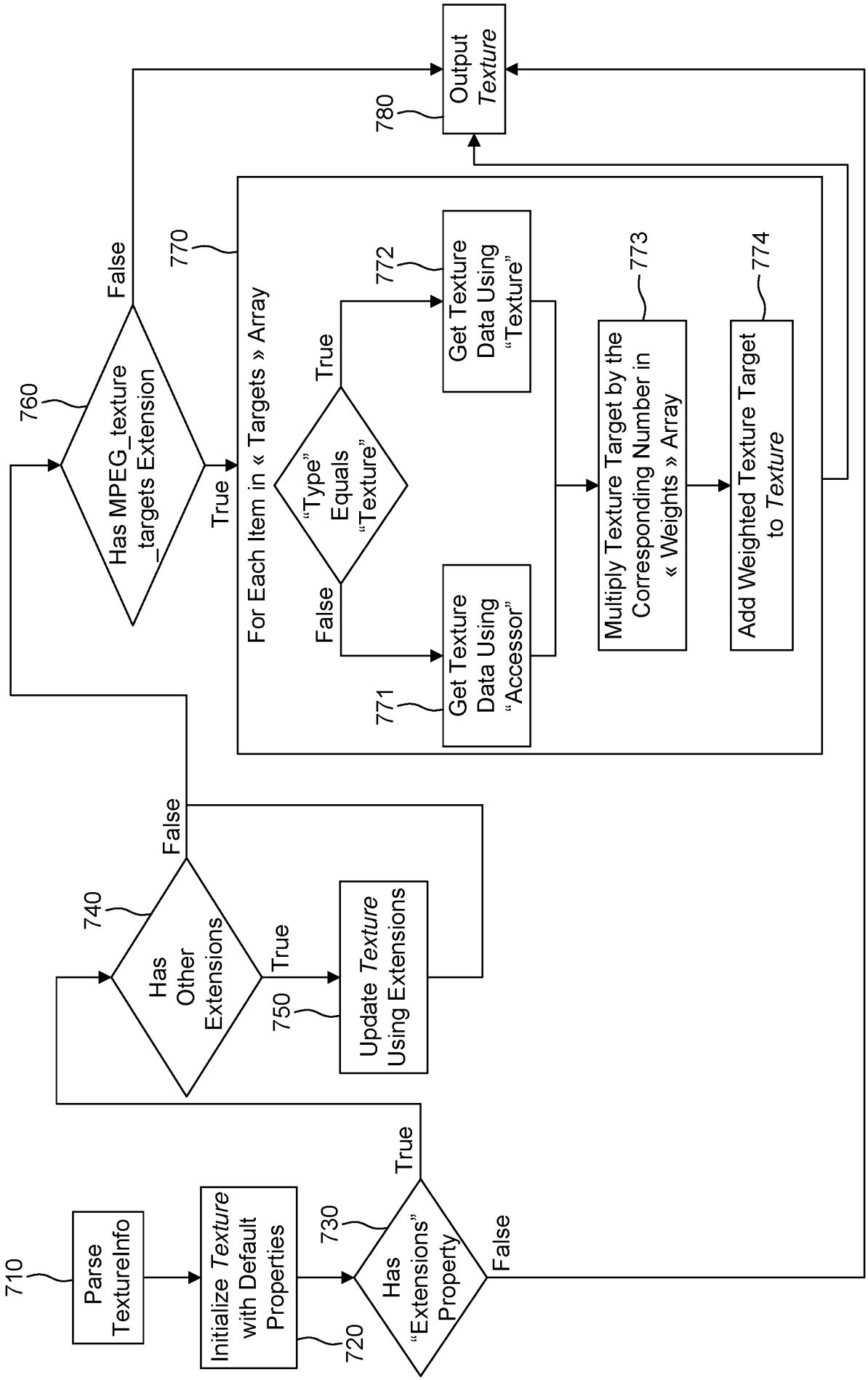

710
Parse TextureInfo
720
Initialize Texture with Default Properties
730
Has "Extensions" Property
True
False
740
Has Other Extensions
True
False
750
Update Texture Using Extensions
760
Has MPEG_texture_targets Extension
True
False
770
For Each Item in « Targets » Array
"Type" Equals "Texture"
True
False
771
Get Texture Data Using "Accessor"
772
Get Texture Data Using "Texture"
773
Multiply Texture Target by the Corresponding Number in « Weights » Array
774
Add Weighted Texture Target to Texture
780
Output Texture

FIG. 7

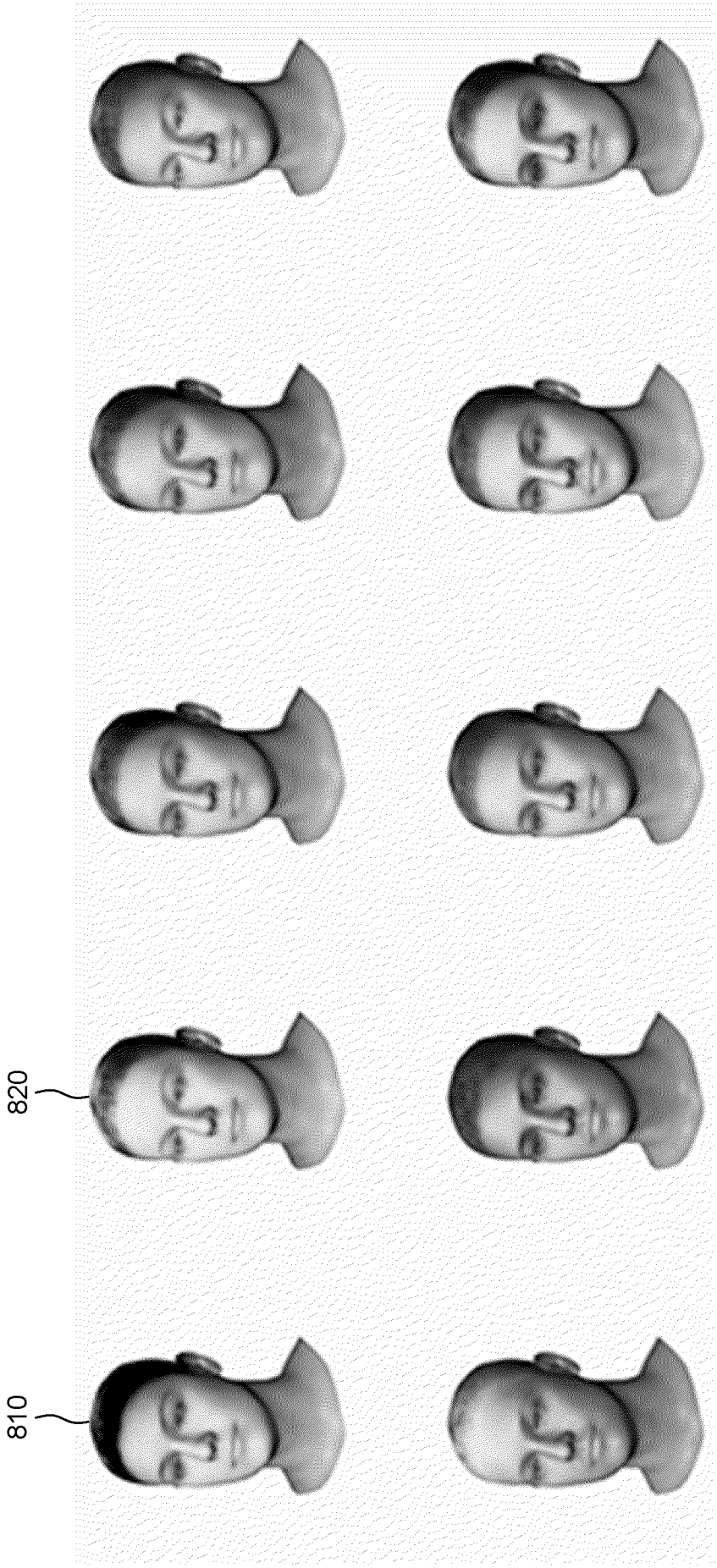


FIG. 8

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2024/065958

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06T15/04 G06T9/00 H04N19/597
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06T H04N

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	"SGIS detail texture-The Detail Texture Extension", INTERNET CITATION, 1 January 1998 (1998-01-01), XP002188570, Retrieved from the Internet: URL:http://techpubs.sgi.com:80/library/manuals/2000/007-2392-002/pdf/007-2392-002.pdf [retrieved on 2002-01-29] pages 129-135 page 144	1-15

-/--

[x] Further documents are listed in the continuation of Box C.

[x] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance;; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance;; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
10 September 2024	18/09/2024

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Santos Luque, Rocio

4

Form PCT/ISA/210 (second sheet) (April 2005)

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	SCHUSTER KERSTEN ET AL: "Compression and rendering of textured point clouds via sparse coding", PROCEEDINGS OF THE 36TH IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, ACMPUB27, NEW YORK, NY, USA, 6 July 2021 (2021-07-06), pages 61-73, XP058907087, DOI: 10.2312/HPG.20211284 ISBN: 978-1-4503-9442-0 figure 2 section 3.1 abstract -----	1-5, 7-12,14, 15
X	US 2005/248582 A1 (SCHEEPERS FERDI [US] ET AL) 10 November 2005 (2005-11-10) figure 4c paragraphs [0069] - [0078] -----	1-5, 7-12,14, 15
X	"Information technology - Runtime 3D asset delivery format - Khronos glTF(TM) 2.0", ISO/IEC 12113:2022, IEC, 3, RUE DE VAREMBÉ, PO BOX 131, CH-1211 GENEVA 20, SWITZERLAND , 26 July 2022 (2022-07-26), pages 1-189, XP082074220, Retrieved from the Internet: URL:https://api.iec.ch/harmonized/publications/download/3038945 [retrieved on 2022-07-26] section 3.9.2; page 47 - page 49 -----	1-4, 6-11, 13-15
A	ALEXANDRE HARDY ET AL: "Blend maps", IT RESEARCH IN DEVELOPING COUNTRIES, SOUTH AFRICAN INSTITUTE FOR COMPUTER SCIENTISTS AND INFORMATION TECHNOLOGISTS, P. O. BOX 392 UNISA 0003 REPUBLIC OF SOUTH AFRICA, 9 October 2006 (2006-10-09), pages 61-70, XP058095012, DOI: 10.1145/1216262.1216269 ISBN: 978-1-59593-567-0 sections 4-5 -----	1-15

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2024/065958

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2005248582 A1	10-11-2005	AU 2004319516 A1	24-11-2005
		CA 2565600 A1	24-11-2005
		EP 1745441 A2	24-01-2007
		NZ 551124 A	31-07-2009
		US 2005248582 A1	10-11-2005
		US 2006022991 A1	02-02-2006
		WO 2005111984 A2	24-11-2005