Image based Object/Environment Representation for VR and MAR

SC 24 WG 9

Seoul, Korea

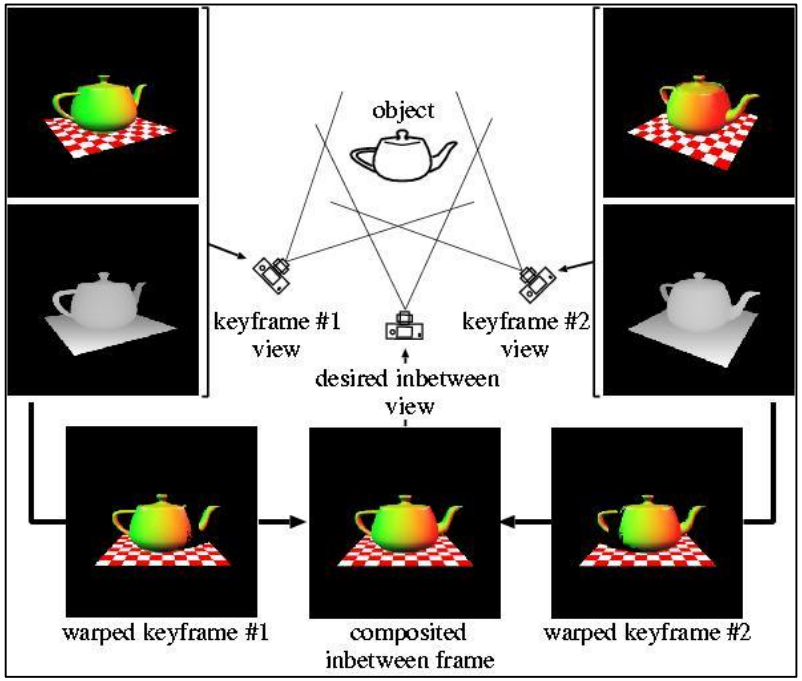
January, 2019

Changhyun Jun, Gerard J. Kim

Korea University

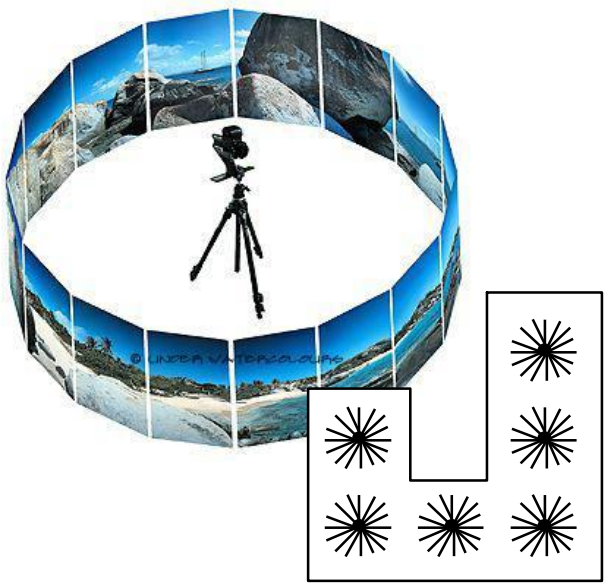
Image based Rendering

- Rely on a set of two-dimensional images of a scene taken from various view points and then render novel/arbitrary views of this scene by view interpolation
- Many variant approaches, but generally:
  - Computationally demanding
    - Requires pixel level processing (e.g. blending, warping, …) of multiple images
      → used to be slow
    - But now it is possible with GPUs and advanced, but relatively low cost hardware
    - "Hole" problems
    - Difficulties with correnspondences
    - Closely related to image based modeling
      - 3D geometry, if can be reconstructed, can be of help in correct image based rendering

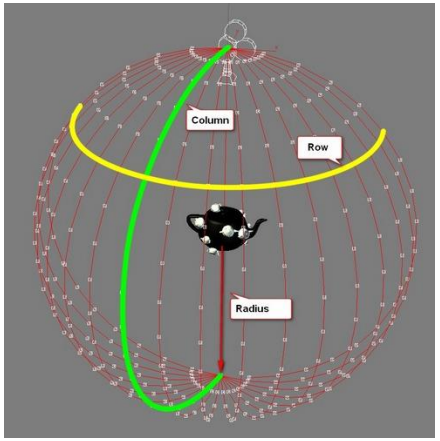


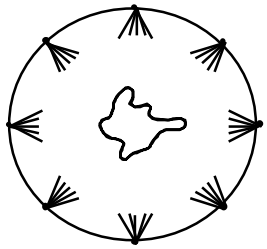
Apple QuickTime VR

- Outward-looking
  - Panoramic views taken at regularly spaced points
  - Each panorama is stitched from an image



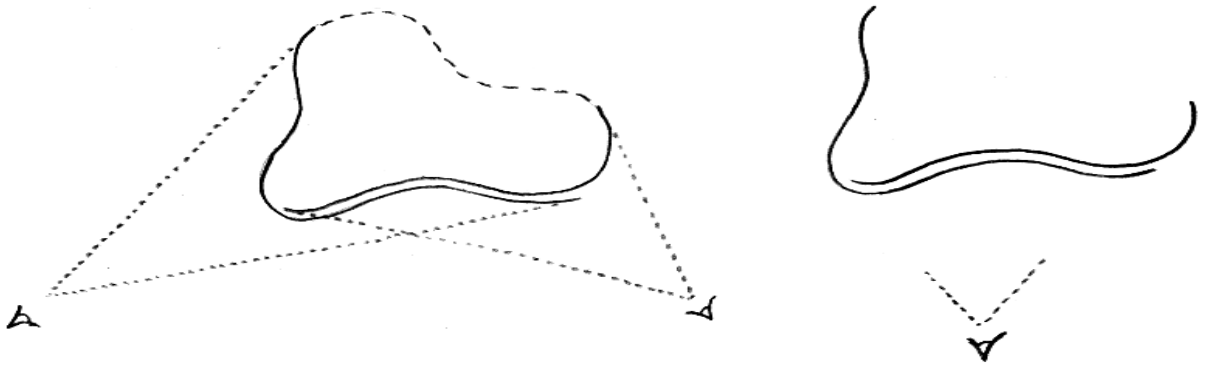
- Inward-looking
  - views taken at points on the surface of a sphere



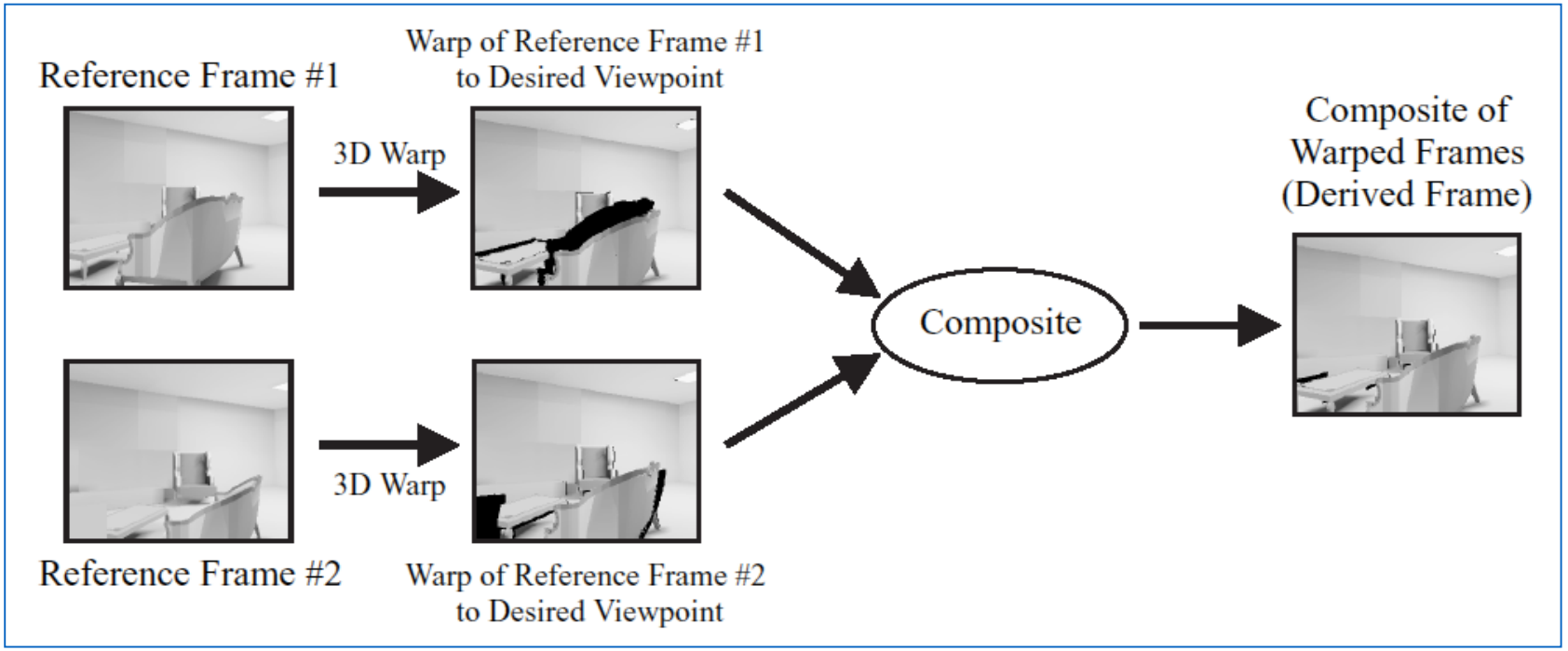


View interpolation / from multiple views

1. Render object from multiple viewpoints
2. (Convert Z-buffers to range images)
3. Tesselate to create multiple meshes
4. Re-render from new viewpoint
5. Use depths to resolve overlaps
6. Use multiple views to fill in holes

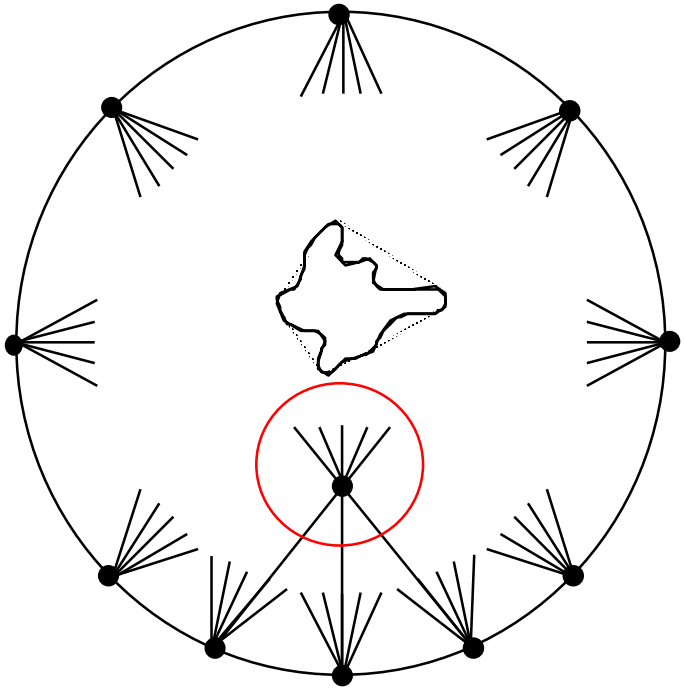


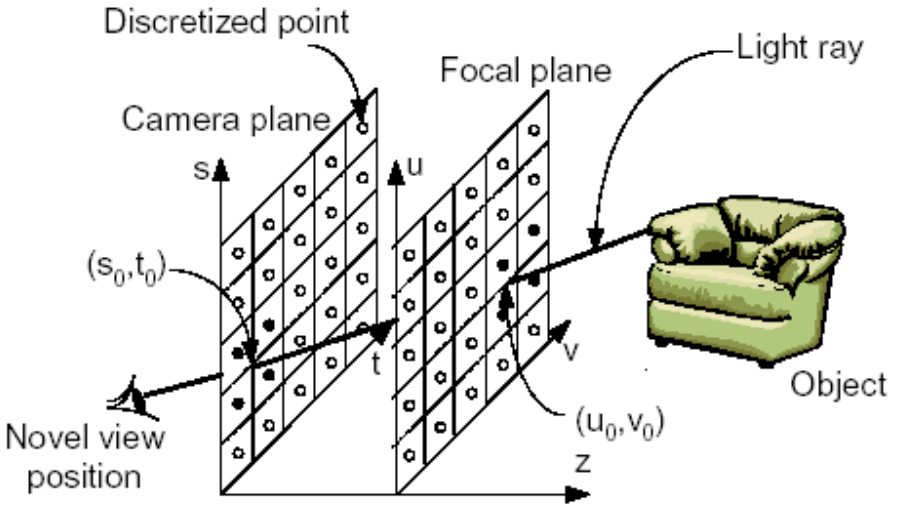
Post-rendering 3D warping



- Convert reference images to polygon meshes (using depth)
- Warp meshes to interpolated viewpoint
- Composite by Z-buffer comparison and conditional write

Light field rendering





- must stay outside convex hull of the object
- like rebinning in computed tomography

The Plenoptic function

Radiance as a function of position and direction in a static scene with fixed illumination

- For general scenes

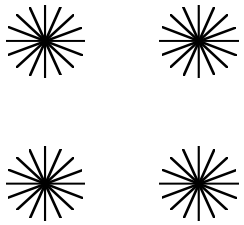
  ⇒ 5D function

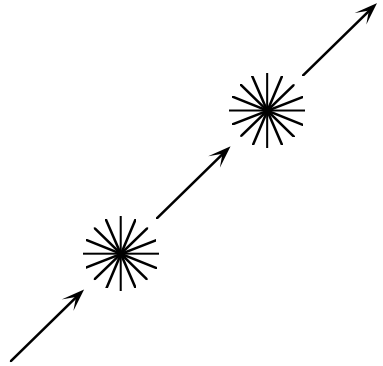
  L (x, y, z, θ, ϕ)

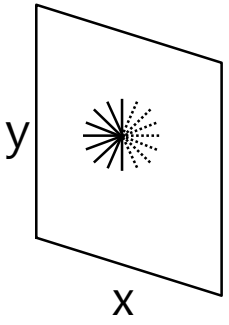
- In free space

  ⇒ 4D function

  " the (scalar) light field"

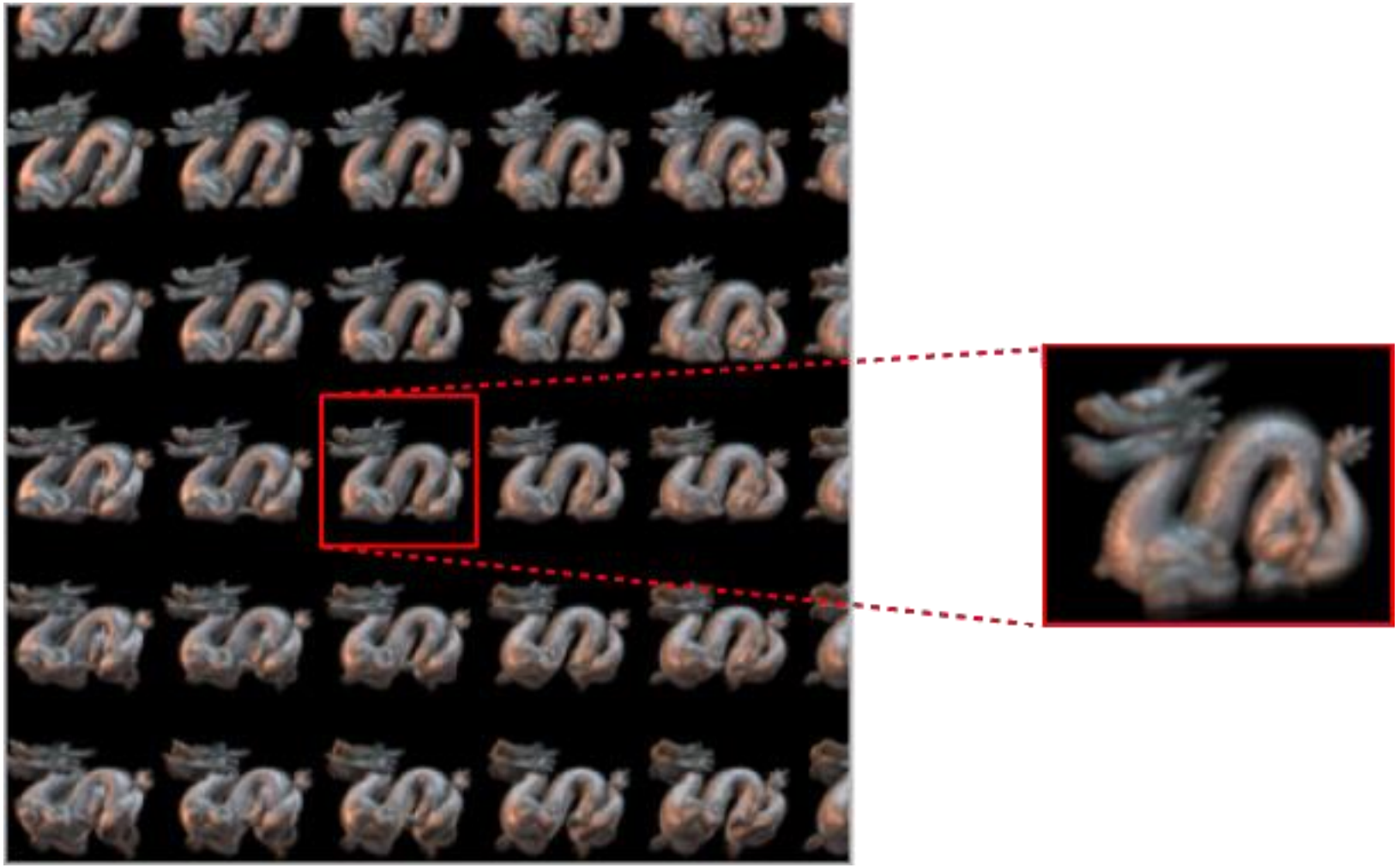






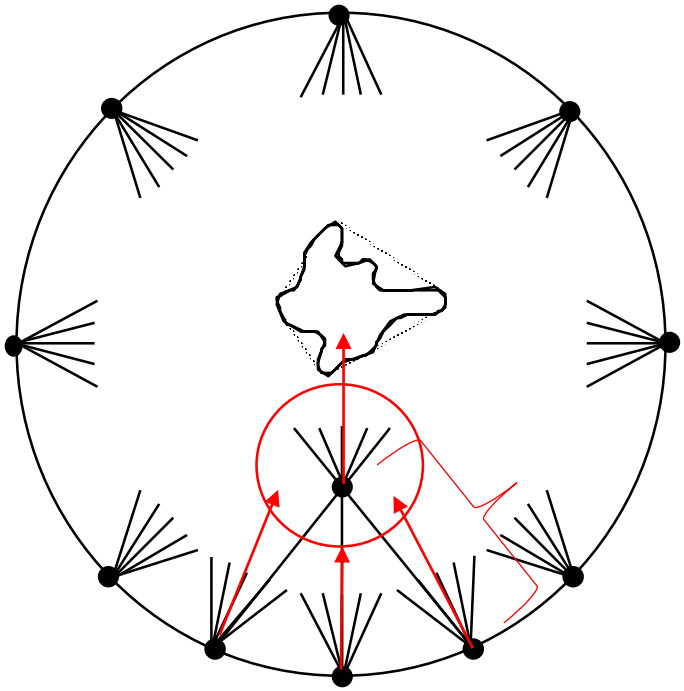
A light field is an array of images

Reconstruct the **plenoptic function** from a set of examples images



What do we need to describe a light field?

- Set of images indexed by view position and view direction
- Each image must optionally be specified/supplied with its camera parameters (for flexible rendering purpose)
- Specification of blending policy
  - Which images and how to mix/interpolate given user location and view direction
    - Angle difference
    - Distance
    - Field of view / Eccentricity
    - 3D information?
    - Relative weight



Conventional IBR

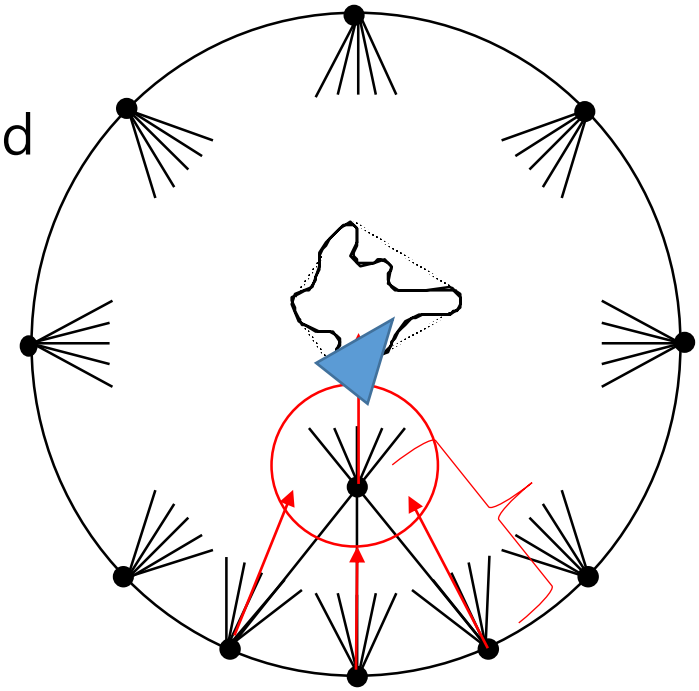
- Image only
  - Low resolution
  - Insufficient information to visualize 3D world for VR/MAR
    - Lack of 3D geometry information

→

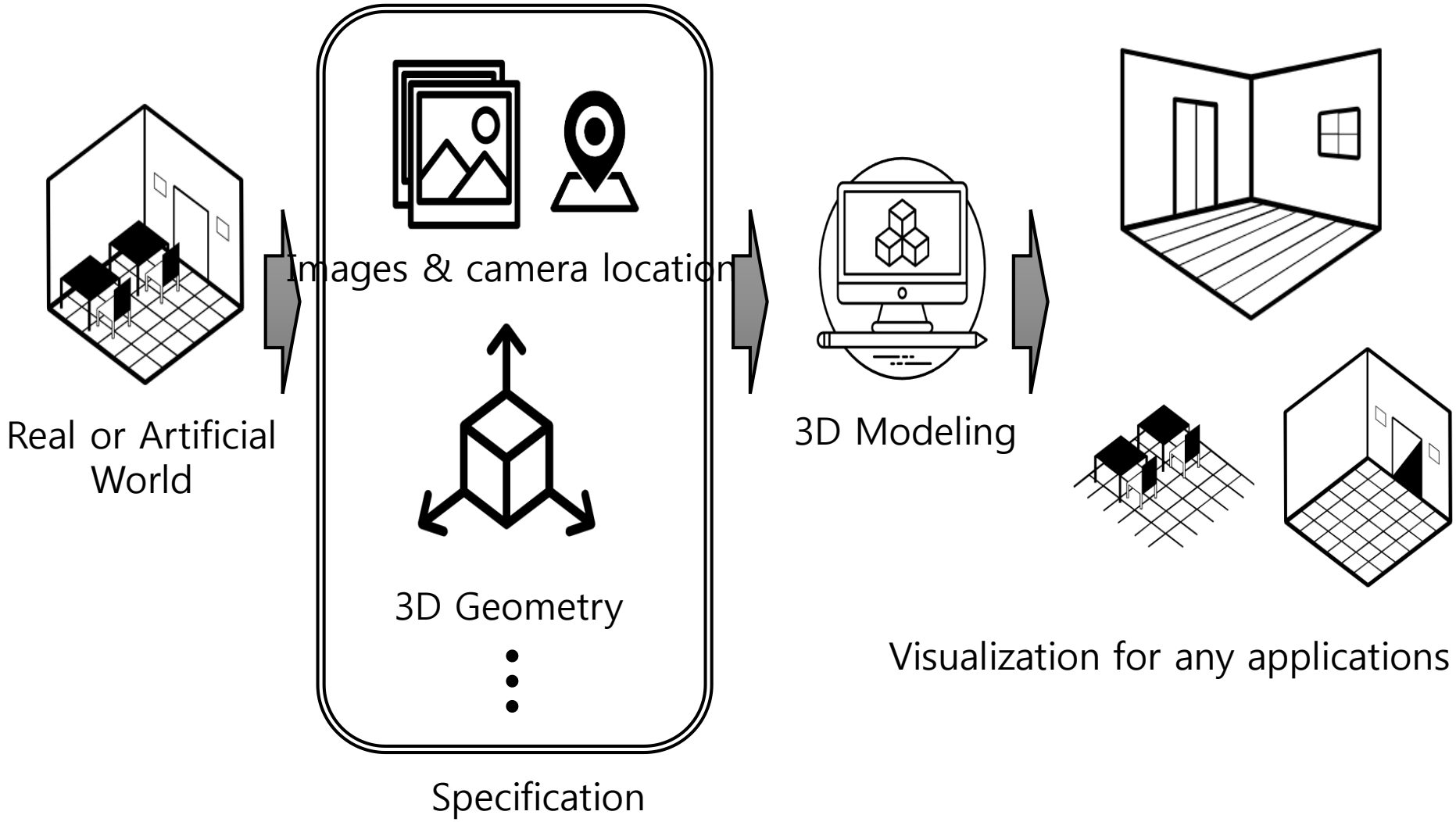
- IBR (+ 3D model)
  - Photo-realistic and accurate real geometry
  - Coverage: Small ~ Large scale

Hybrid environment: IBR (or Light field) + 3D model

- Set of images indexed by view position and view direction
- Each image must optionally be specified/supplied with its camera parameters (for flexible rendering purpose)
- Specification of blending policy
- 3D model on to which images are mapped
  - Textured indexed face list
    - Image to vertex / face / polygon mapping



Proposed specification



E.g. X3D nodes? – Indexed face list, Multiple texture

<IndexedFaceSet DEF='PumpHouseIFS'

coordIndex='0 1 5 4 -1 5 1 2 6 -1 6 2 3 7 -1 3 0 4 7 -1 1 12 13 2 -1 2 13 14
-1 12 15 16 13 -1 15 0 3 16 -1 16 3 17 -1 9 5 6 10 -1 8 9 10 11 -1 4 8 11 7
-1 4 5 9 8 -1 11 10 6 7 -1 3 2 14 17 -1 13 16 17 14 -1'
>

16 quads

<Coordinate

point='0.0 0.0 0.0 2.0 0.0 0.0 2.0 1.75 0.0 0.0 1.75 0.0 0.625 0.75 0.0 1.0 0.75
0.0 1.0 1.6 0.0 0.625 1.6 0.0 0.625 0.75 -0.65 1.0 0.75 -0.65 1.0 1.6 -0.65 0.625
1.6 -0.65 2.0 0.0 -2.7 2.0 1.75 -2.7 2.0 2.5 -1.0 0.0 0.0 -2.7 0.0 1.75 -2.7 0.0
2.5 -1.0'
/>

17 vertices

</IndexedFaceSet>

E.g. X3D nodes? – Indexed face list, Multiple texture

```
ImageTexture : X3DTexture2DNode, X3DUrlObject {
SFNode     [in,out]     metadata NULL [X3DMetadataObject]
MFString   [in,out]     url [] [URI]
SFBool     []           repeatS TRUE
SFBool     []           repeatT TRUE
SFNode     []           textureProperties NULL [TextureProperties] }
```

Viewpoint?

```
<Shape DEF='MovieShapeStandardDefinition'>
  <IndexedFaceSet DEF='Quadrilateral320x240' coordIndex='0 1 2 3' solid='false'>
    <Coordinate point='-1.6 -1.2 0 1.6 -1.2 0 1.6 1.2 0 -1.6 1.2 0'/>
    <TextureCoordinate DEF='FullImageMapping' point='0 0 1 0 1 1 0 1'/>
  </IndexedFaceSet>
  <Appearance>
    <TextureTransform/>
    <MovieTexture DEF='X3dQuipMovieStandardDefinition'
     repeatS='false' repeatT='false'
     url='"X3dQuipBrutzmanStandardDefinitionMPEG1.mpg" "X3dQuipBrutzmanStandardDefin
  </Appearance>
</Shape>
```

MultiTexture

For each MT set?

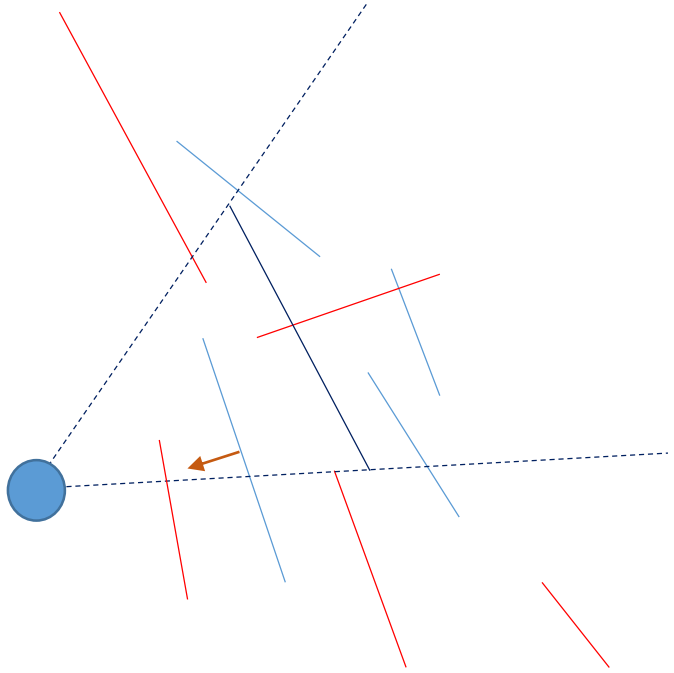
Per shape/IFS

- The MultiTexture node specifies the application of several individual textures to a 3D object to achieve a more complex visual effect. MultiTexture can be used as a value for the texture field in an Appearance node.
- The texture field contains a list of texture nodes (e.g., ImageTexture, PixelTexture, and MovieTexture). The texture field may not contain another MultiTexture node.
- The color and alpha fields define base RGB and alpha values for SELECT mode operations.
- The mode field controls the type of blending operation. The available modes include MODULATE for a lit Appearance, REPLACE for an unlit Appearance, and several variations of the two. The mode field may contain an additional blending mode for the alpha channel.
  - EXAMPLE The mode value '"MODULATE","REPLACE"' specifies Color = (Arg1.color × Arg2.color, Arg1.alpha).
- The number of used texture stages is determined by the length of the texture field. If there are fewer mode values, the default mode is "MODULATE".

MODE	Description
"MODULATE"	Multiply texture color with current color Arg1 × Arg2
"REPLACE"	Replace current color Arg2
"MODULATE2X"	Multiply the components of the arguments, and shift the products to the left 1 bit (effecti vely multiplying them by 2) for brightening.
"MODULATE4X"	Multiply the components of the arguments, and shift the products to the left 2 bits (effec tively multiplying them by 4) for brightening.
"ADD"	Add the components of the arguments Arg1 + Arg2
"ADDSIGNED"	Add the components of the arguments with a -0.5 bias, making the effective range of val ues from −0.5 through 0.5.
"ADDSIGNED2X"	Add the components of the arguments with a -0.5 bias, and shift the products to the left 1 bit.
"SUBTRACT"	Subtract the components of the second argument from those of the first argument. Arg1 − Arg2
"ADDSMOOTH"	Add the first and second arguments, then subtract their product from the sum. Arg1 + Arg2 − Arg1 × Arg2 = Arg1 + (1 − Arg1) × Arg2
"BLENDDIFFUSEALPHA"	Linearly blend this texture stage, using the interpolated alpha from each vertex. Arg1 × (Alpha) + Arg2 × (1 − Alpha)
"BLENDTEXTUREALPHA"	Linearly blend this texture stage, using the alpha from this stage's texture. Arg1 × (Alpha) + Arg2 × (1 − Alpha)
"BLENDFACTORALPHA"	Linearly blend this texture stage, using the alpha factor from the MultiTexture node. Arg1 × (Alpha) + Arg2 × (1 − Alpha)
"BLENDCURRENTALPHA"	Linearly blend this texture stage, using the alpha taken from the previous texture stage. Arg1 × (Alpha) + Arg2 × (1 − Alpha)
...	...

```
<Shape>
  <!-- TODO add a single geometry node here -->
  <IndexedFaceSet>
    <MultiTextureCoordinate>
      <!-- TODO add multiple TextureCoordinate nodes here, match corresponding MultiTexture children -->
      <TextureCoordinate point='0 0 1 0 1 1 0 1'/>
      <TextureCoordinate point='0 0 1 0 1 1 0 1'/>
      <TextureCoordinate point='0 0 1 0 1 1 0 1'/>
    </MultiTextureCoordinate>
  </IndexedFaceSet>
  <Appearance>
    <Material/>
    <MultiTexture alpha='0.8' color='0.9 1 0.2' function='"" "COMPLEMENT" "ALPHAREPLICATE"'
        mode='"MODULATE" "REPLACE" "BLENDDIFFUSEALPHA"' source='"DIFFUSE" "SPECULAR" "FACTOR"'>
      <!-- TODO add a multiple texture nodes here -->
      <ImageTexture/>  <MovieTexture/>  <PixelTexture/>
    </MultiTexture>
    <MultiTextureTransform>
      <!-- TODO add multiple TextureTransform nodes here, match corresponding MultiTexture children -->
      <TextureTransform/>   <TextureTransform/>   <TextureTransform/>
    </MultiTextureTransform>
  </Appearance>
</Shape>
```

Example 1: Pure IBR

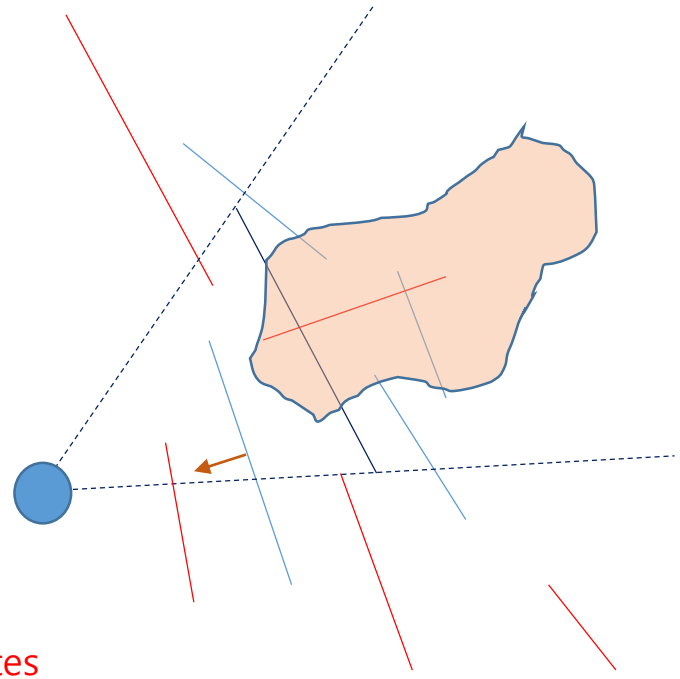


- Just set of indexed/organized textures
  - Augment the current model with
    - Camera information for each texture
      - Use/borrow the viewpoint node model?
    - Provide overall blending policy
      (default is pixel to pixel interpolation)

```
<MultiTexture alpha = … color = … function = … mode = …>
 <ImageTexture url = "x0.jpg"> <CameraInfo … /> </ImageTexture>
 <ImageTexture url = "x1.jpg"> <CameraInfo … /> </ImageTexture>
 …
</MultiTexture>
 …
```

<Viewpoint description='IndexedQuadSet Texture' fieldOfView='3.0307e-01' orientation='1.000 0.000 0.000 1.571' position='0.0000e+00 -1.6371e+01 1.0025e-15'/>

Example 2: Hybrid



- Set of indexed/organized textures

  + Index Face Set

    - Augment the current model with
      - Camera information for each texture
      - Provide overall blending policy

        (default is pixel to pixel interpolation)
      - Associating each indexed vertex to texture coordinates

        in a more compact way for a large model

```
<shape>
 <IndexedFaceSet coordIndex='0 1 2 3 -1 4 5 6 7 -1 5 6 7 -1 …">
  <Coordinate point = 'x1 y1 z1 x2 y2 z2 …/>
  <TextureCoordinate point = '0 0 1 0 1 1 0 1 …/>
  <TextureCoordinate point = '0 0 1 0 1 1 0 1 …/>
  …
 </IndexedFaceSet>

 <MultiTexture alpha = … color = … function = … mode = …>
  <ImageTexture url = "x0.jpg"> <CameraInfo … /> </ImageTexture>
  <ImageTexture url = "x1.jpg"> <CameraInfo … /> </ImageTexture>
  …
 </MultiTexture>
  …
```

<Shape>

<IndexedQuadSet index='0 30 31 1 30 60 61 31 60 90 91 61 90 120 121 91 120 150 151 121 150 180 181 151 180 210 211 181 210 240 241 211 240 270 271 241 270 300 301 271 300 330 331 301 330 360 361 331 360 390 391 361 390 420 421 391 420 450 451 421 450 480 481 451 480 510 511 481 510 540 541 ... solid='false' ccw='true' colorPerVertex='true' normalPerVertex='true'>

<Coordinate point='-1.6730 0.4483 -1.0000 -1.7229 0.4616 -0.9048 -1.7677 0.4736 -0.8069 -1.8073 0.4843 -0.7066 -1.8416 0.4934 -0.6043 -1.8705 0.5012 -0.5002 -1.8939 0.50
0.2879 -1.9240 0.5155 -0.1803 -1.9306 0.5173 -0.0722 -1.9315 0.5176 0.036
1.9164 0.5135 ... />

<Normal vector='0.8365 -0.2241 0.5000 0.8614 -0.2308 0.4524 0.8838 -0.23
0.3533 0.9208 -0.2467 0.3021 0.9352 -0.2506 0.2501 0.9469 -0.2537 0.1973
0.9620 -0.2578 0.0902 0.9653 -0.2587 0.0361 0.9658 -0.2588 -0.0181 0.963
0.2568 -0.1260 0.9502 -0.2546 -0.1796 0.9394 -0.2517 -0.2326 0.9259 -0.24
0.3364 0.8907 -0.2387 -0.3868 0.8692 -0.2329 -0.4362 0.8451 -0.2264 -0.48
0.7896 -0.2116 -0.5760 0.7583 -0.2032 -0.6194 0.7248 -0.1942 -0.6610 0.6892 -0.1847 -0.7007 0.6515 -0.1746 -0.7383 0.6119 ... />

<TextureCoordinate point='0.0000 0.0000 0.0000 0.0345 0.0000 0.0690 0.0000 0.1034 0.0000 0.1379 0.0000 0.1724 0.0000 0.2069 0.0000 0.2414 0.0000 0.2759 0.0000 0.3103 0.0000 0.3448 0.0000 0.3793 0.0000 0.4138 0.0000 0.4483 0.0000 0.4828 0.0000 0.5172 0.0000 0.5517 0.0000 0.5862 0.0000 0.6207 0.0000 0.6552 0.0000 0.6897 0.0000 0.7241 0.0000 0.7586 0.0000 0.7931 0.0000 0.8276 0.0000 0.8621 0.0000 0.8966 0.0000

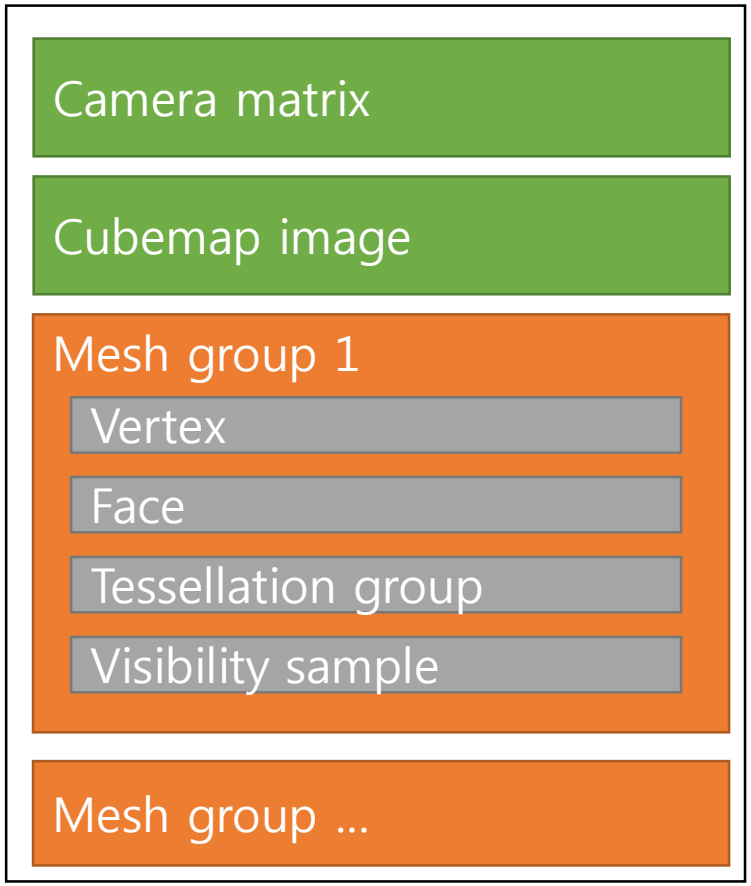
http://www.web3d.org/x3d/content/examples/Basic/CAD/IndexedQuadSetTextureIndex.html

But …

- Light field images are not exactly textures … (in concept)
  - Can be used as textures, if we had 3D environment models
- Proposal (c.f. Real world reconstruction)
  - Create a structure for a separate light field description
  - Include more elaborate indexing mechanism
  - Addendum to MAR content information model
  - Syntax – using X3D models and XML

General structure with simple example

- Simple example

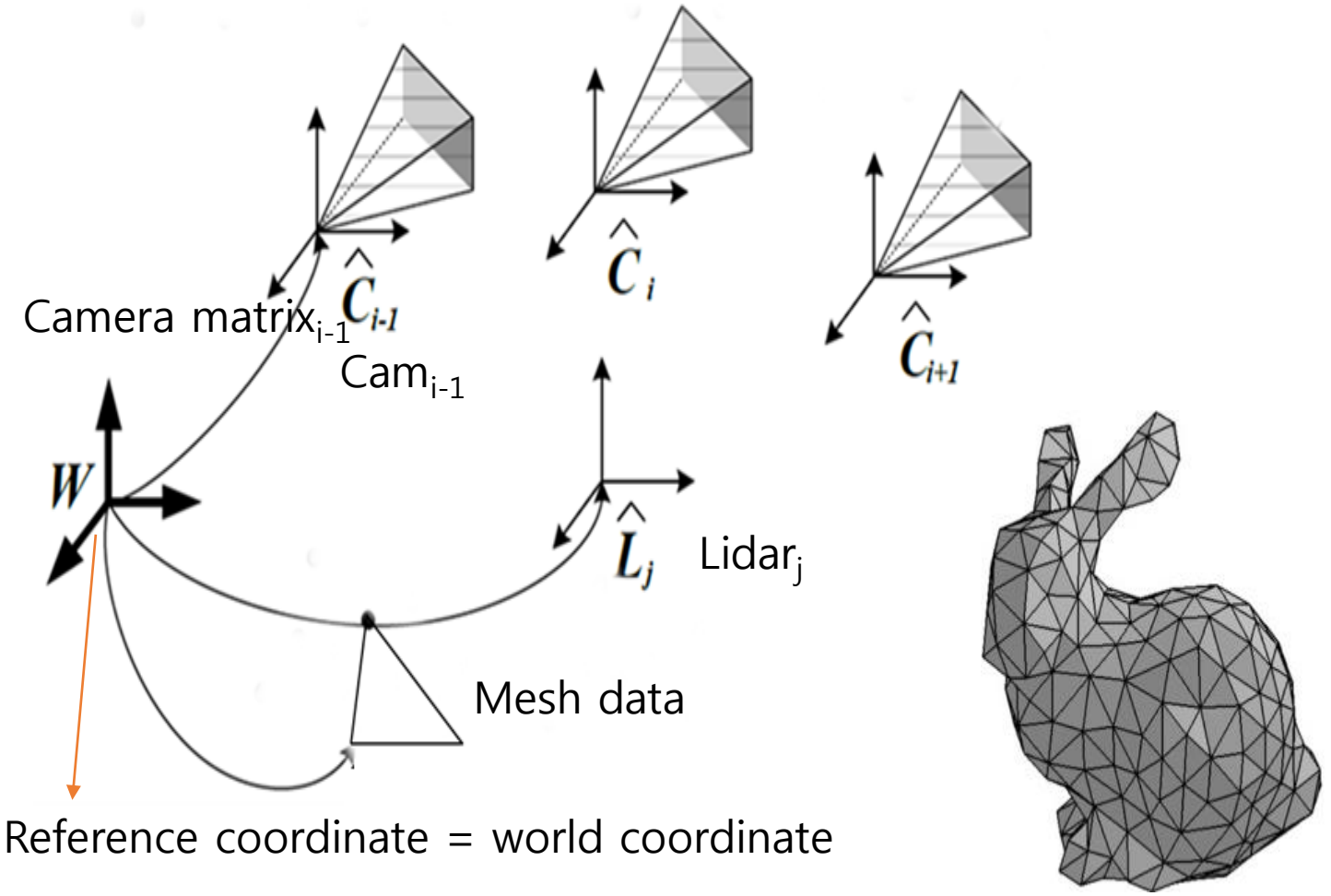



```
# camera matrix
# world space
# (c) (#) (4x4matrix)
c 0     -0.255893 0.962670 0.088234 -1.119945   -0.966661 -0.253944 -0.032854 -0.596478 -
0.009220 -0.093700 0.995558 -0.090753    0.000000 0.000000 0.000000 1.000000
c 1     -0.260595 0.961200 0.090460 -2.250560   -0.965403 -0.258536 -0.034007 -0.558877 -
0.009300 -0.096192 0.995319 0.003159    0.000000 0.000000 0.000000 1.000000
c 2     -0.280730 0.955609 0.089451 -3.866500   -0.959765 -0.278883 -0.032780 -0.457779 -
0.006378 -0.095054 0.995451 0.150382    0.000000 0.000000 0.000000 1.000000

# cubemap image
# (i) (#) (path)
i 0 181218_KIST1F_m3_r2/cube_scene0000004.jpg
i 1 181218_KIST1F_m3_r2/cube_scene0000009.jpg
i 2 181218_KIST1F_m3_r2/cube_scene0000016.jpg


# mesh group
# (g) (group name)
g group1

# vertex
# world space
# (v) (x) (y) (z)
v 1.477429 11.484982 0.832000
v 1.499517 11.487240 -1.071000
v 1.655002 11.502548 -1.070000
v 1.682921 11.504905 0.832000
v 1.559104 10.718448 0.832000
v 1.746606 10.737210 0.832000
v -2.256639 7.318503 2.182240
v -2.218855 7.394930 2.182130
v -2.552782 6.973721 -0.687350
v -2.540890 6.988780 0.132520
...

# face
# (f) (index0) (index1) (index2)
f 0 2840 2841
f 2 2842 2843
f 855 5 856
f 0 857 858
f 11 2844 2845
...


# tesellation group
# a,b,c = barycentric coordinate
# (t) (sample num) (a) (b) ... (a) (b) ...|
t 19    1 0 0 1 0 0     0.875000 0.062500       0.625000 0.312500       0.625000 0.187500
0.625000 0.062500       0.375000 0.562500       0.375000 0.437500       0.375000 0.312500
0.375000 0.187500       0.375000 0.062500       0.125000 0.812500       0.125000 0.687500
0.125000 0.562500       0.125000 0.437500       0.125000 0.312500       0.125000 0.187500
0.125000 0.062500

# visibility sample
# (s) (cam0) (visibility) (cam1) (visibility) ...
s 0     0 11111111111111111111   1 11111111111111111111   2 11111111111111111111
s 1     0 11111111111111111111   1 11111111111111111111   2 11111111111111111111
s 2     0 00000000000000000000   1 00000000000000000000   2 00000000000000000000
s 3     0 11111111111111111111   1 11111111111111111111   2 11111111111111111111
```

General structure with simple example

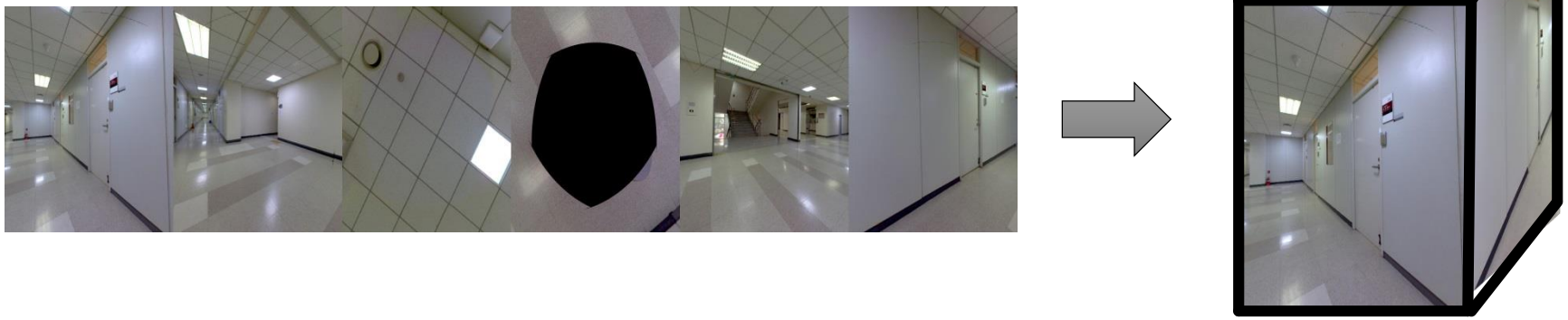
- Camera matrix
  - This attribute contains the homogeneous transformation matrix (4 by 4) with respect to world coordinate
  - Format: (c) (#) (4x4matrix)
  - Start with 'c' : identifier of a camera matrix attribute
  - #: the index of a camera
  - 4x4matrix : the homogeneous transformation matrix of the camera

General structure with simple example

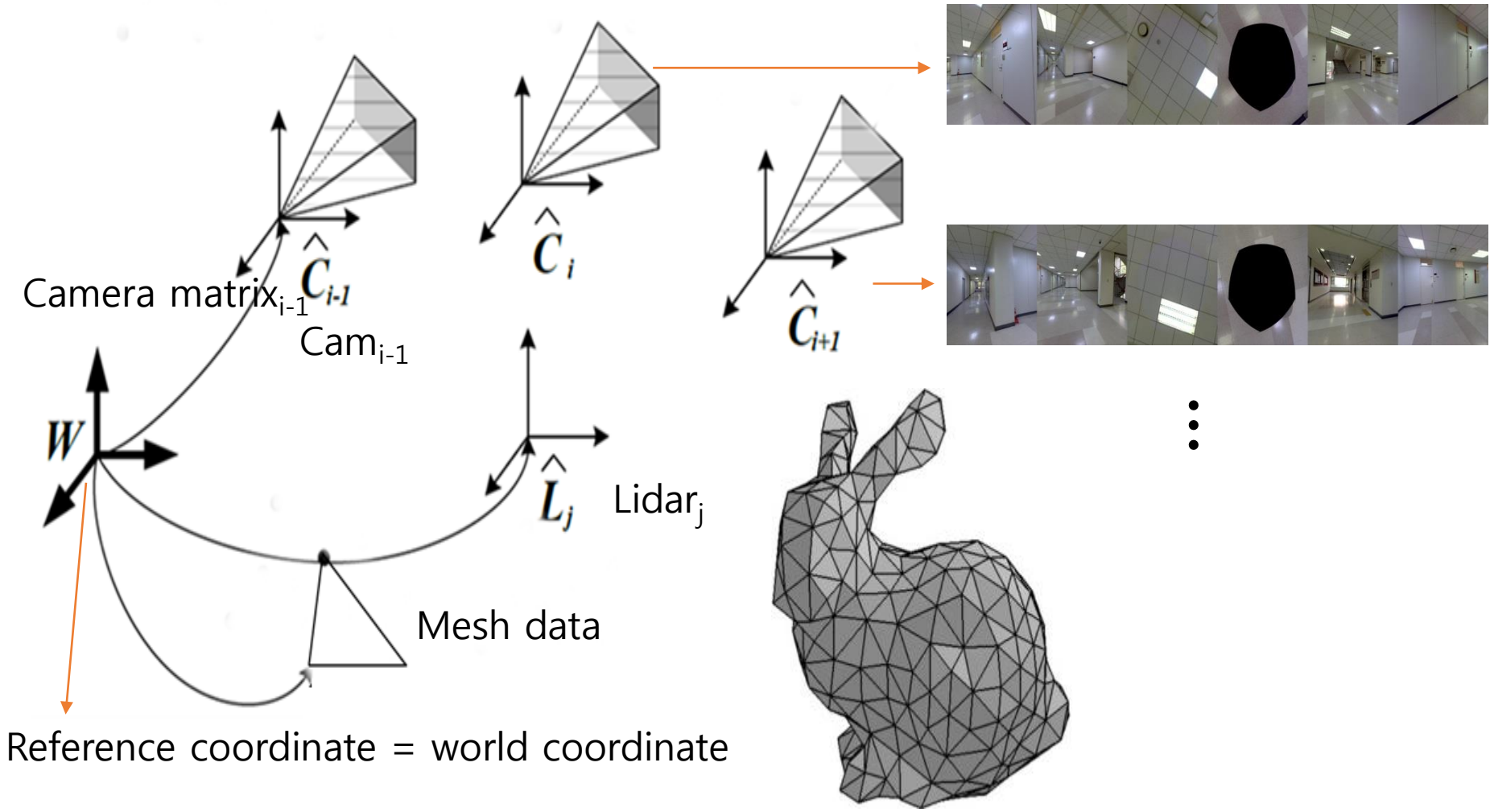


General structure with simple example

- Cubemap image
  - This attribute contains a local path of a cubemap image file
  - Format: (i) (#) (path)
  - Start with 'i' : identifier of a cubemap image attribute
  - #: the index of a cubemap image file
  - path : the local path of a file



General structure with simple example



General structure with simple example

- Mesh group
  - This attribute contains mesh data for a geometric model
  - Each mesh group contains vertex, face, tessellation group, and visibility sample attributes for mesh data
  - Format: (g) (group name)
  - Start with ‘g’ : identifier of a mesh group attribute
  - Group name: the name of a mesh group.

General structure with simple example

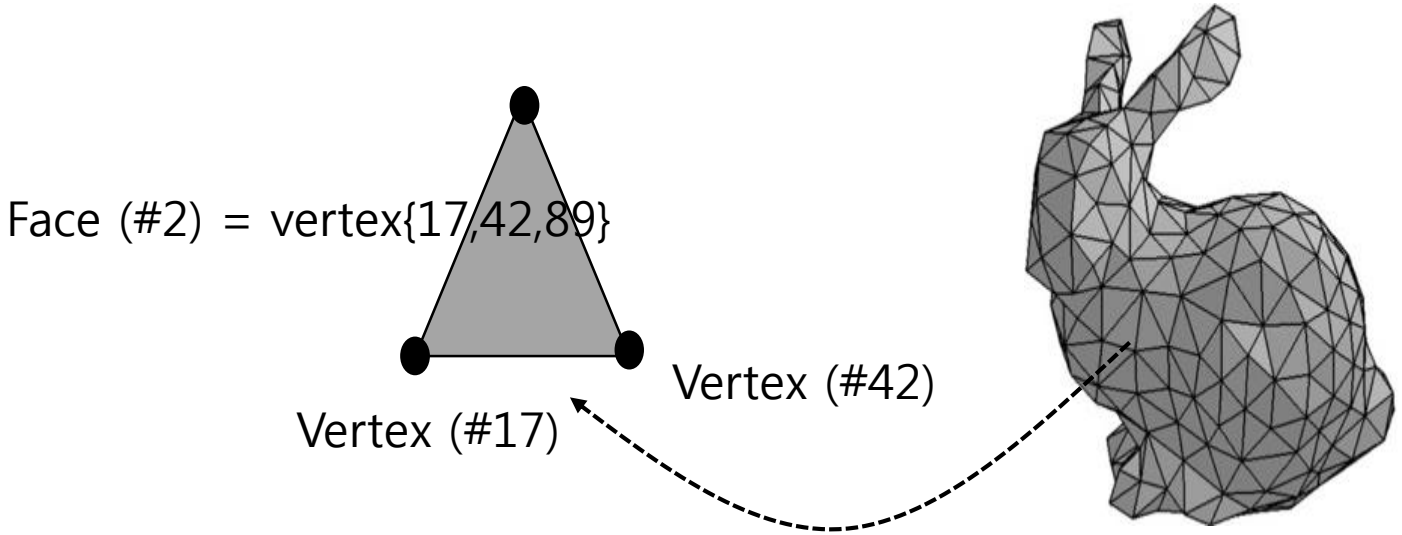
- vertex
  - This attribute is sub-attribute of a mesh group.
  - It contains vertices data of a mesh w.r.t. world coordinate.
  - Unit: meter
  - Each row has one vertex data of mesh
  - Format: (v) (x) (y) (z)
  - Start with ‘v’ : identifier of a vertex attribute
  - x : x-axis value of the vertex
  - y : y-axis value of the vertex
  - z : z-axis value of the vertex

General structure with simple example

- face
  - This attribute is sub-attribute of a mesh group.
  - It contains triangle faces data of a mesh
  - Each row has one face data of mesh
  - Format: (f) (index0) (index1) (index2)
  - Start with ‘f’ : identifier of a face attribute
  - Index0, index1, index2: indices of vertices

General structure with simple example

Vertex (#89) = {x=0.1454, y=1.2985, z=0.0954}

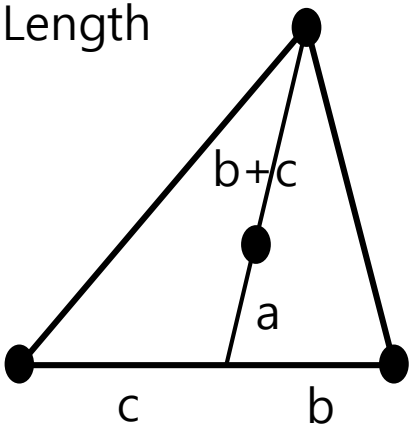


General structure with simple example

- Tessellation group
  - This attribute is a sub-attribute of a mesh group.
  - It contains a sampling size and its coordinate values for sub-sampling.
  - Format: (t) (sample num) (a) (b) … (a) (b) …
  - Start with ‘t’ : identifier of a tessellation group attribute
  - sample num: sampling size
  - a, b: homogeneous barycentric coordinate values of each sample

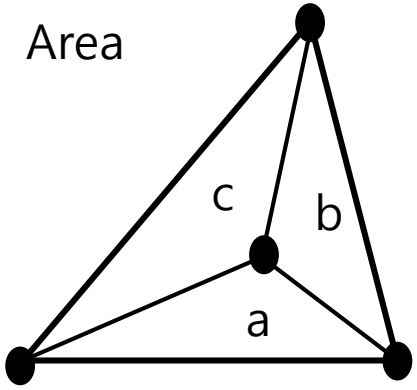
General structure with simple example

Length



homogeneous barycentric coordinate, a+b+c=1

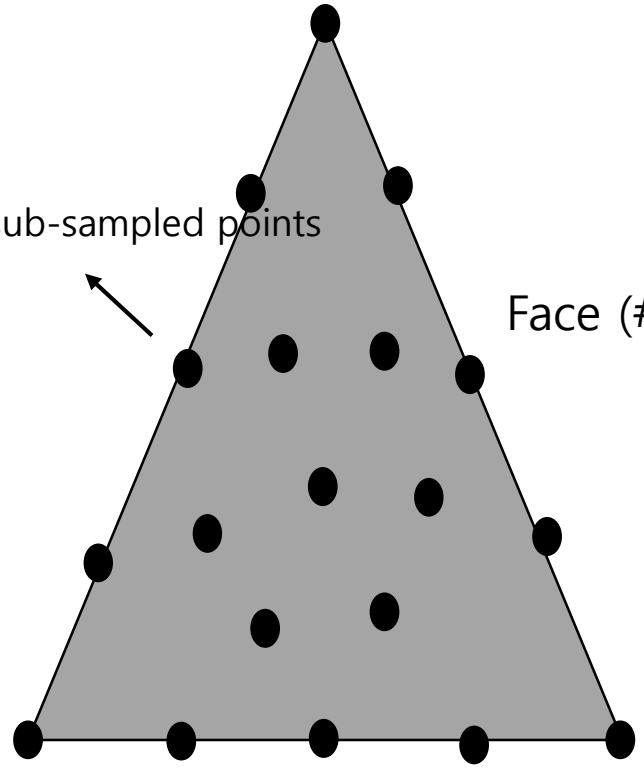
Area



General structure with simple example

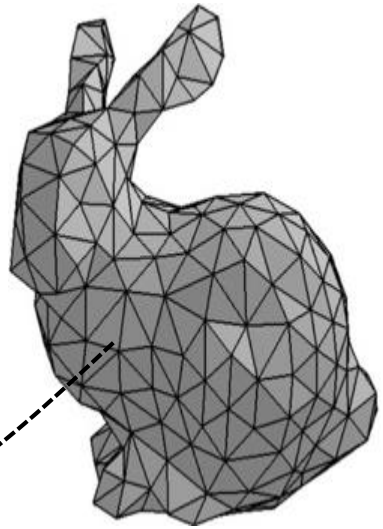
- visibility sample
  - This attribute is sub-attribute of a mesh group.
  - It contains a visibility of sub-sampled region of each face w.r.t. all camera locations.
  - Each row has visibilities of the corresponding face w.r.t. all camera locations.
  - Format: (s) (#) (cam0) (visibility) (cam1) (visibility) ...
  - Start with 's' : identifier of a visibility sample attribute
  - #: the index of the corresponding face
  - cam# : the index of the corresponding camera location
  - Visibility: it contains 0 or 1. The number of '0' or '1' is as same as the sampling size of tessellation group
    - '0' = the sub-sampled face is invisible w.r.t. the camera location
    - '1' = the sub-sampled face is visible w.r.t. the camera location

General structure with simple example



Each face has sub-sampled points

Face (#2)



s 2 0 00000000000000000000 1 00000000000000000000

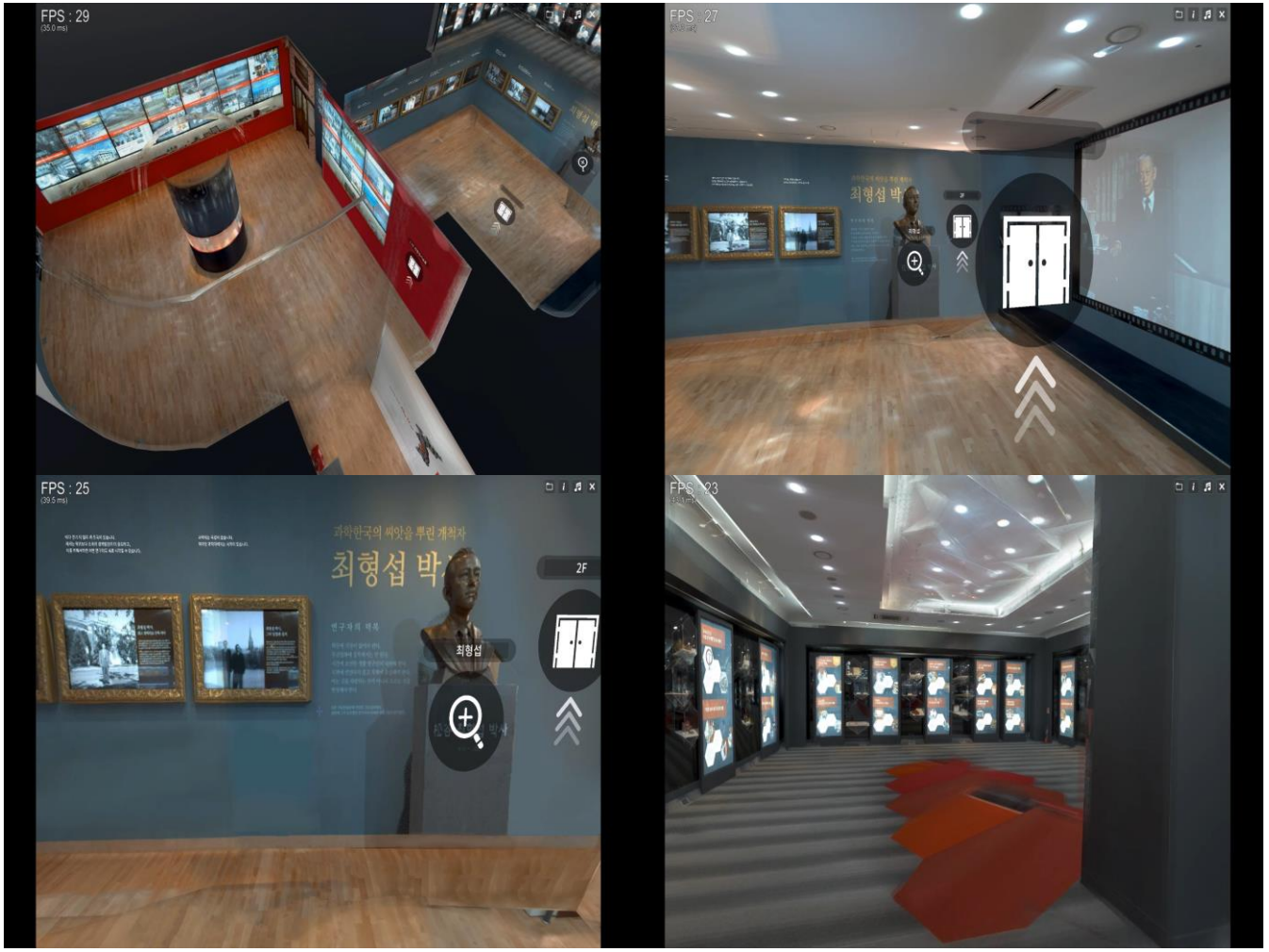
2 11111111111111111111

Index of face data Index of camera

6th's sub-sampled point is invisible from camera 1

3rd's sub-sampled point is visible from

Proposed specification



Proposed specification