ELSEVIER SCIENCE

An Effective Method for Specifying Projective Texture Mapping in X3D

In-Kwon Kim[a], Add (Brutzman, Puk), Jong-Sung Ha[c], Nakhoon Baek[e], Kwan-Hee Yoo[a,b,*]

[a]*Dept. of Digital Information Convergence, Chungbuk National University*

[b]*Dept. of Computer Science, Chungbuk National University*

[c]*Dept. of Game and Contents, Wooksuk University*

[e] *School of Computer Sci. and Eng., Kyungpook National University*

Elsevier use only: Received date here; revised date here; accepted date here

Abstract

This paper presents the specification and implementation of a new functionality called projective texture mapping (PTM) in the Extensible 3D (X3D) that is the ISO standard for defining 3D interactive web-contents. PTM is a method projecting an image, also known as the texture image, onto a scene as if by a slide projector, which is useful for enhancing image quality in a variety of lighting techniques. Even though there have been attempts to specify PTM in X3D, users need to iteratively specify images and coordinates of textures on each object in a 3D scene. As a standard functionality of X3D, we suggest effective PTM methods to provide perspective and parallel projective textures, and the implementation results of PTM rendering in an open source X3D viewer called FreeWRL are shown.

Keywords: Projective texture mapping; X3D; scene graph

1. Introduction

The *Extensible 3D* (X3D) [1,2] is a royalty-free ISO standard file format for representing interactive 3D contents in the world-wide-web, based on the *Extensible Markup Language* (XML). This paper is concerned with developing a new texture mapping functionality called *Projective Texture Mapping* (PTM) in X3D. PTM is a specialized texture mapping method, which was first introduced by Everitt [3]. PTM actually allows a texture image to be projected onto objects as if projected by a slide projector. This technique is useful for various applications such as photo formation [4,5] as well as for enhancing image quality in rendering [6,7,8]. X3D represents the 3D virtual world with the scene graph that is a general data structure commonly used in the area of 3D computer graphics and Web environment. A scene graph is the ordered collection of fundamental components called *node* in a graph or tree structure. A node may have many children, with the effect of the parent node applied to all its children nodes.

Even though X3D includes various nodes for texturing features, none of them has supported PTM yet. Kamburelis [9] suggested a PTM method by using the technique of shadow mapping of Everitt [6,7] with additive PTM information. Kim *et. al* [10] also tried to support PTM with different methods defining an independent node for PTM information for Appearance node. These methods are inconvenient and exhaustive because users have to repeatedly define nodes for PTM information for each of objects in a scene. In other words, PTM is processed *in local* for each object in the scene.

In order to resolve the inconvenience, this paper suggests an effective method in order to *globally* process

PTM with respect to all objects at once in a scene. And we also define two types of PTM nodes: *perspective* and *parallel* PTM nodes, still strictly obeying the existing hierarchy of X3D node structures. The two types of proposed PTM nodes have been implemented in X3D viewer of FreeWRL [11] by using OpenGL shader languages [12]. The implementation details and those results are presented in Section 4.

We will start with previous works in Section 2. The details of the PTM node design were presented in Section 3. Experimental results are followed in Section 4. Finally conclusions and future work are followed.

2. Related Works

In this section, we show a set of works directly related to PTM methods. Concepts and theoretical details are presented.

2.1. PTM Concept

PTM is a technique to project a texture image onto the surfaces of objects within the projection volume with a view from a certain spot called 3D scene projection point, as illustrated in Fig. 1. The projection volume is determined by projection parameters such projection point, direction, and aspect ratio, which was described in the process of PTM by Everitt [3].

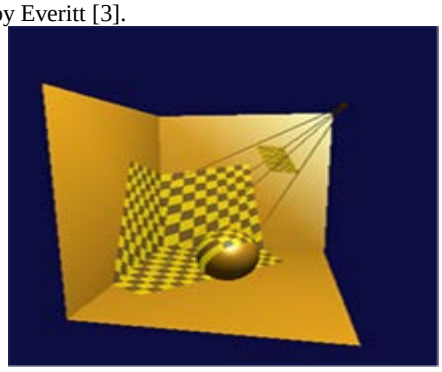

Fig. 1. Projective Texture Mapping[10]

In the classical texture mapping techniques, 2D texture coordinate values with the range from 0 to 1 are mapped onto the vertices of 3D models. In order to understand PTM concept, we consider the *object linear* method in which a texture is fixed in an object space. The method computes the texture coordinate, $T^t = (s,t,r,q)$ for a vertex, $V^t = (x_0, y_0, z_0, w_0)$, by multiplying a concatenated matrix T_0 into the vertex, V^t, as shown in the following Equations:

$$\begin{bmatrix} s \\ t \\ r \\ q \end{bmatrix} = T_0 \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ w_0 \end{bmatrix} \quad (1)$$

with

$$T_0 = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} P_p V_p M \quad (2)$$

where P_p , V_p , and M are transformation matrices for projection, camera view, and model, respectively.

However, the concept does not reflect one of projective texture mapping which allows a texture image to be projected onto the scene as if by a slide projector. Since the texture to be projected is fixed in an eye space with viewpoint, the method called *eye linear*, is used to the texture coordinates on an object in PTM. First, object coordinates are transformed into the eye space by the following equation:

$$\begin{bmatrix} x_e \\ y_e \\ z_e \\ w_e \end{bmatrix} = \begin{bmatrix} Camera \\ \text{View} \\ \text{Matrix} \end{bmatrix} \begin{bmatrix} Modeling \\ \text{Matrix} \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ w_0 \end{bmatrix} \quad (3)$$

Then, a matrix called the eye linear generator is multiplied as follows:

$$\begin{bmatrix} s \\ t \\ r \\ q \end{bmatrix} = T_s \begin{bmatrix} X_e \\ Y_e \\ Z_e \\ W_e \end{bmatrix} \times \begin{bmatrix} Invers \\ eye \\ View \\ Mat \end{bmatrix} \quad (4)$$

The eye linear generator matrix T_s can be approximated as the following Equation:

$$T_s = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} P_p V_p V_e^{-1}, \quad (5)$$

based on a projector instead of a camera in order to represent the projector's properties.

2.2. Previous PTM extensions with local effects

Kamburelis obtained PTM coordinates by extending the existing *X3DLightNode* with new fields describing the minimum and maximum distance of the projection and the up-vector of the light, as in Table 1 [9]. The obtained coordinates are used in the PTM of *ProjectedTextureCoordinate* node inheriting the existing *X3DTextureCoordinateNode* as in Table 2.

Since this method has the structure dividing the nodes into *ProjectedTextureCoordinate* and *ImageTexture*, PTM necessarily requires the interaction among the nodes of *X3DLight*, *ProjectedTextureCoordinate*, and *ImageTexture*.

Furthermore, users have to iteratively define the two nodes of *ProjectedTextureCoordinate* and *ImageTexture* for each of objects in a virtual scene.

Table 1. New fields in X3DLightNode.

SFFloat	[in,out]	projectionNear	0
SFFloat	[in,out]	projectionFar	0
SFBool	[in,out]	up	0 0 0
SFBool	[in,out]	defaultShadowMap	NULL

Table 2. ProjectedTextureCoordinate Node

ProjectedTextureCoordinate : X3DTextureCoordinateNode			
SFNode	[in,out]	projector	NULL
# [SpotLight, DirectionalLight, X3DViewpointNode]			

Recently, Kim *et al.* [10] tried to resolve the limitations of Kamburelis's method by defining a new node *ProjectiveTexture* as in Table 3. This method generates an independent projector node *PerspectiveProjector* as in Table 4, and divides the two nodes into *X3DLight* and *PerspectiveProjector* to obtain the PTM coordinates, while Kamburelis's method exploits the node *X3DLight*.

Still the inconvenient problem has been also remained that users have to iteratively define two other nodes of *ProjectiveTexture* and *Appearance* for each of objects in a scene.

Table 3. ProjectedTextureCoordinate Node

ProjectedTextureCoordinate : X3DTextureNode {			
SFNode	[in,out]	metadata	Null
		[X3DMetadataObject]	
SFString	[in,out]	projectorName	""
SFBool	[in,out]	value	true
MFString	[in,out]	url	""
}			

Table 4. PerspectiveProjector Node

PerspectiveProjector : X3DChildNode {			
SFNode	[in,out]	metadata	Null
		[X3DMetadataObject]	
SFString	[in,out]	description	""
SFVec3f	[in,out]	centerOfProjection	0 0 0 (-∞,∞)
SFVec3f	[in,out]	direction	0 0 1 (-∞,∞)
SFFloat	[in,out]	fieldOfView	45
SFFloat	[in,out]	aspectRatio	1
SFFloat	[in,out]	nearFar	1 10
SFVec3f	[in,out]	upVector	0 1 0
}			

3. A New PTM Extension with Global Effects

Similar to the X3DLight node in the X3D node structure hierarchy, a PTM node needs to be defined without any violation of the original hierarchy in order to globally effect all objects in a scene at once. First, we define a new node TextureProjectorNode inheriting ChildNode as like the LightNode obtaining the global effect by inheriting the ChildNode, as illustrated in Fig. 2. Next, two other new nodes TextureProjectorPerspective and TextureProjectorParallel are defined by inheriting TextureProjectorNode that have the effects of perspective and parallel projection, respectively.

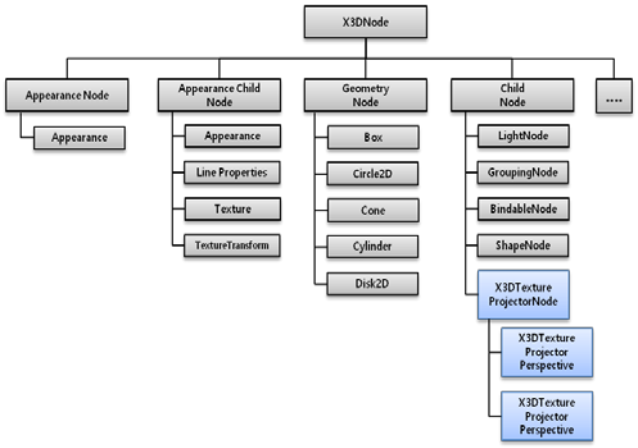

Fig. 2. New Nodes in the Hierarchy of X3D Node Structure

Table 5, 6, and 7 describe each definition of three node *TextureProjectorNode*, *TextureProjectorPerspective*, and *TextureProjectorParallel*, in which the last columns describe the default value of each field. *TextureProjectorNode* inherits X3DChildNode of the original X3D hierarchy, and again other two nodes *TextureProjectorPerspective* and *TextureProjectorParallel* inherits *TextureProjectorNode*.

In Table 5, the field "description" is a value identifying projective texture projectors arranged in 3D space. The projectors are located in the position of "location" with the viewing direction of "direction". The field "aspectRatio" of the projectors cannot be given by users, but can be obtained from other fields. The fields of "nearDistance" and "nearDistance" represent the minimum and maximum distances that can be projected, respectively. The fields of "global" and "on" are defined to represents the characteristics of children nodes; if "global" is true and "on"

is false, the projector node effects only its children nodes, but it effects all objects within the projected volume if "global" is false and "on" is true. The 2D image to be projected is described in the field "texture".

Table 5. X3DTextureProjectorNode

X3DTextureProjectorNode:X3DChildNode {			
SFNode	[in,out]	metadata [X3DMetadataObject]	Null
SFString	[in,out]	description	""
SFVec3f	[in,out]	location	0 0 0 (-∞,∞)
SFVec3f	[in,out]	direction	0 0 1 (-∞,∞)
SFFloat	[in,out]	aspectRatio	
SFFloat	[in,out]	nearDistance	1
SFFloat	[in,out]	farDistance	10
SFBool	[in,out]	global	
SFBool	[in,out]	on	
SFNode	[in,out]	texture [X3DTexture2DNode]	
}			

In Table 6, two more fields are added in order to describe the properties of perspective projection; "fieldOfView" is the view angle of the projector, and "upVector" is the angle rotating along Z-axis.

Table 6. : X3DTextureProjectorPerspective Node

X3DTextureProjectorPerspective:X3DTextureProjectorNode {			
SFFloat	[in,out]	fieldOfView	π/4 (0,π)
SFFloat	[in,out]	aspectRatio	
}			

Table 7, only a simple field is added in order to describe the properties of parallel projection; "fieldOfView" is the X-, Y-coordinate values of the screen.

Table 7. X3DTextureProjectorParallel Node

X3DTextureProjectorParallel:X3DTextureProjectorNode {			
SFFloat	[in,out]	fieldOfView	(-1,1,-1,1) (-∞,∞)
}			

4. Implementation and Experiments

We implemented the rendering of newly defined PTM nodes with OpenGL shader language (GLSL) in FreeWRL,2015 that is an open source X3D viewer. The system environments for implementing and testing the projective texture mapping of X3D is described in Table 8.

Table 8. System Environments

Software	OS	MS Windows 7 (64 bit version)
	Compiler	MS Visual Studio 2008
	3D Graphics API	OpenGL Shader Language
	X3D Viewer	FreeWRL (C Language)
Hardware	CPU	Intel(R) Core(TM) i7-4702MQ CPU @ 2.20GHz
	Memory	12GB
	GPU	NVIDIA GeForce GT 740M

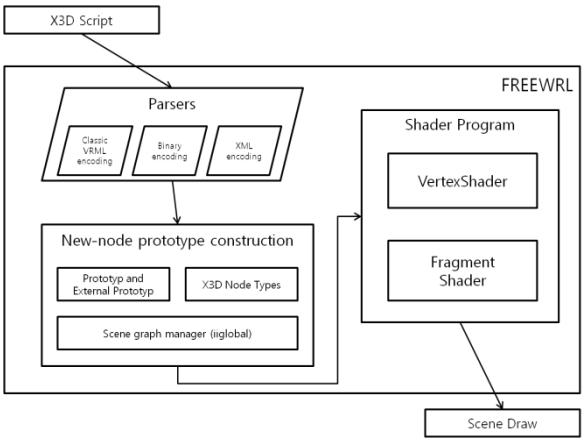

Fig. 3. Pipeline Architecture of FreeWRL

Fig. 3 illustrates the pipeline architecture of FreeWRL, in which, each node of X3D script files are parsed to produce a tree hierarchical structure, and then rendered by shader programs. As illustrated in Appendix A, FreeWRl can be extended with PERL programming to parse the new nodes: *X3DTextureProjectorPerspective* and *X3DTextureProjectorParallel*.

Table 9 is the implementation of OpenGL shader programs for rendering of the projective texture mapping in X3D. The uniform variable "projTexGenMatCam0" is the eye linear generator matrix generated in the parsing of nodes. The computed texture coordinate in the varying variable "projTexCoord" is passed to the next stage. The

condition “projTexCoord.q > 0.0” checks the projection from the reverse of viewing direction.

Table 9. Shader Programs for Projective Texture Mapping

Vertex Shader

```
uniform mat4 projTexGenMatCam0;
uniform mat4 viewMat;
varying vec4 projTexCoord;
void vertProjCalTexCoordinate(void) {
   mat  invViewMat = invers(viewMat);
   vec4 posEye = gl_ModelviewMatrix *gl_Vertex;
   vec4 posWorld = invViewMat *pos_Eye;
   projTexCoord = projTexGenMatCam0 *posWorld;
};
```

Fragment Shader

```
varying vec4 projTexCoord;
vec4 projMapColor_forCam1;
void fragProjCalTexCoord(void) {
   if (projTexCoord.q > 0.0) {
      projMapColor_forCam1
      = texture2Dproj(fw_Texture_unit0, projTexCoord);
   }
}
```

Table 10-13 shows the examples of X3D scripts using our PTM nodes, and their rendered results are shown in Fig. 4-7, respectively.

Table 10. Perspective Project of an Apple Image into a Plane

```
<X3D profile="Interactive" version="3.3">
<Scene>
<TextureProjectorPerspective
            description='pt1' location='3 3 3' direction='-1 0 -1'
            fieldOfView='15' nearDistance='1' farDistance='10'
            upVector='0 1 0' global= 'true' on= 'true'>
            <ImageTexture url='C:/image/apple.jpg' repeatS='false'
repeatT='false'/>
</TextureProjectorPerspective>
<Shape>
   <Appearance>
            <Material diffuseColor='0.5 0.5 0.5'/>
   </Appearance>
   <IndexedFaceSet solid='false' coordIndex="3 2 1 0 -1, 4 5 2 3-1,
5 6 1 2 -1">
      <Coordinate point="1 0 1, -1 0 1, -1 0 -1, 1 0 -1, 1 1 -1, -1 1 -
1, -1 1 1 "/>
   </IndexedFaceSet>
</Shape>
</Scene>
</X3D>
```

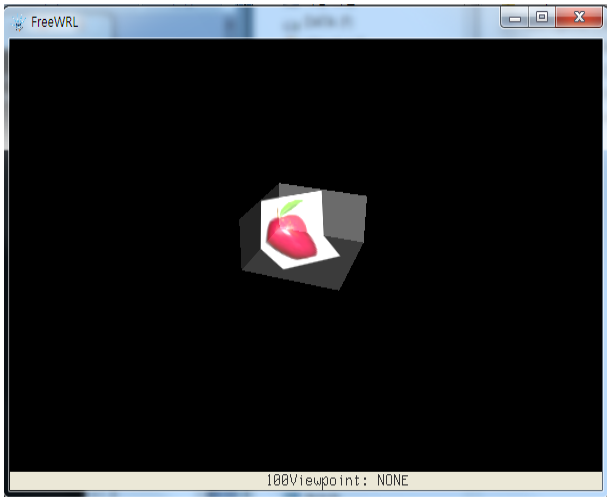

Fig. 4. Rendered Result of Table 10

Table 11. Perspective Project of an Apple Image into a Box

```
<X3D profile="Interactive" version="3.3">
<Scene>
<TextureProjectorPerspective
            description='pt1' location='3 3 3' direction='-1 -1 -1'
            fieldOfView='15' nearDistance='1' farDistance='10'
            upVector='0 1 0' global= 'true' on= 'true'>
            <ImageTexture url='C:/image/apple.jpg' repeatS='false'
repeatT='false'/>
</TextureProjectorPerspective>
<Shape>
   <Appearance>
            <Material diffuseColor='0.5 0.5 0.5'/>
   </Appearance>
   <Box/>
</Shape>
</Scene>
</X3D>
```

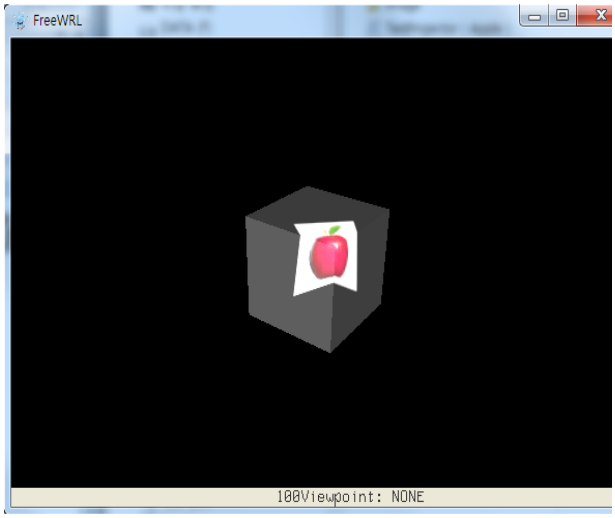

Fig. 5. Rendered Result of Table 11

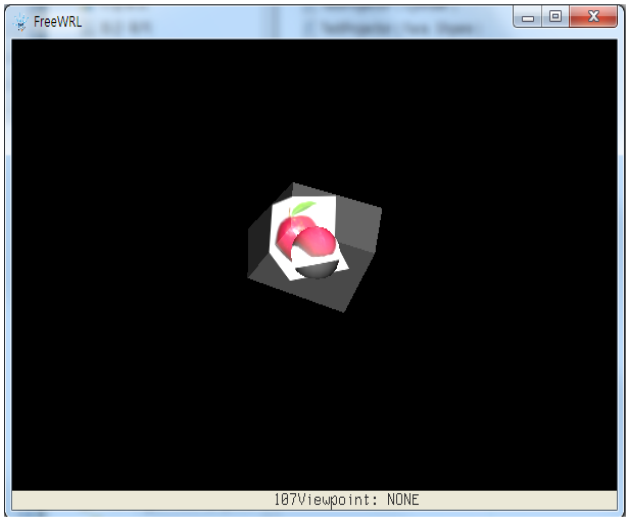

Fig. 6. Rendered Result of Table 12

Table 12. Perspective Project of an Apple Image into a Sphere and a Plane

```
<X3D profile="Interactive" version="3.3">
<Scene>
<TextureProjectorPerspective
            description='pt1' location='3 3 3' direction='-1 0 -1'
            fieldOfView='15' nearDistance='1' farDistance='10'
            upVector='0 1 0' global= 'true' on= 'true'>
            <ImageTexture url='C:/image/apple.jpg' repeatS='false'
repeatT='false'/>
</TextureProjectorPerspective>
<Shape>
   <Appearance>
            <Material diffuseColor='0.5 0.5 0.5'/>
   </Appearance>

   <IndexedFaceSet solid='false' coordIndex="3 2 1 0 -1, 4 5 2 3-1,
5 6 1 2 -1">
      <Coordinate point="1 0 1, -1 0 1, -1 0 -1, 1 0 -1, 1 1 -1, -1 1 -
1, -1 1 1 "/>
   </IndexedFaceSet>
</Shape>
<Transform translation='0,0.25,0'>
<Shape>
   <Appearance>
            <Material diffuseColor='0.5 0.5 0.5'/>
   </Appearance>
   <Sphere radius = '0.5'/>
</Shape>
</Transform>
</Scene>
</X3D>
```

Table 13. Perspective Project of an Apple Image into two Planes

```
<X3D profile="Interactive" version="3.3">
<Scene>
<TextureProjectorPerspective
            description='pt1' location='3 3 3' direction='-1 0 -1'
            fieldOfView='15' nearDistance='1' farDistance='10'
            upVector='0 1 0' global= 'true' on= 'true'>
            <ImageTexture url='C:/image/apple.jpg' repeatS='false'
repeatT='false'/>
</TextureProjectorPerspective>
<Shape>
   <Appearance>
            <Material diffuseColor='0.5 0.5 0.5'/>
   </Appearance>
   <IndexedFaceSet solid='false' coordIndex="3 2 1 0 -1, 4 5 2 3-1,
5 6 1 2 -1">
      <Coordinate point="1 0 1, -1 0 1, -1 0 -1, 1 0 -1, 1 1 -1, -1 1 -
1, -1 1 1 "/>
   </IndexedFaceSet>
</Shape>
<Transform translation='0,0.25,0'>
<Shape>
   <Appearance>
            <Material diffuseColor='0.5 0.5 0.5'/>
   </Appearance>
   <Sphere radius = '0.5'/>
</Shape>
</Transform>
</Scene>
</X3D>
```

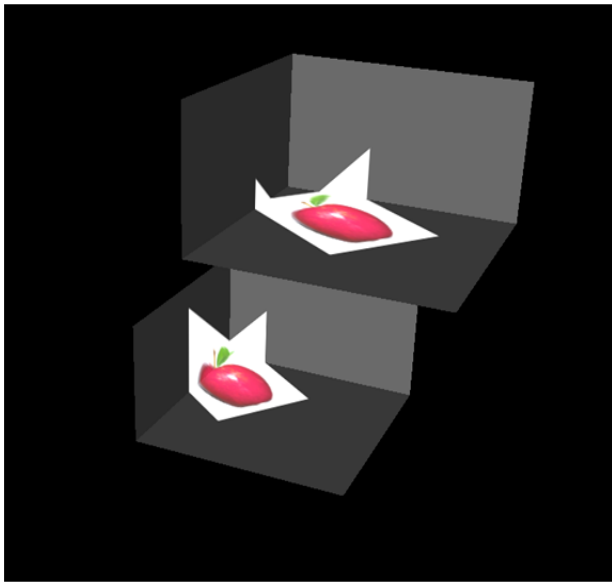

Fig. 7. Rendered Result of Table 13

5. Conclusion

In this paper, we provide two new X3D nodes, to support projective texture mapping (PTM), in the web contents. These PTM nodes can be naturally integrated into the exiting hierarchy of X3D, through extending the parser capability of X3D viewers such as of FreeWRL 2015 and implementing the rendering shader programs. Since these PTM nodes globally effect all objects in the scene at once, we could improve the efficiency in the previous solutions of [9] and [10], through separately creating two nodes per each appearance node of multiple objects.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2014R1A1A2055379) and by Woosuk University

References

[1] Web3D Consortium Specifications of “VRML”, “X3D”, “X3D texture”, and “X3D 3D Texturing Component” http://web3d.org/x3d/specifications, June 2015.
[2] Don Brutzman and Leonard Daly, “X3D: Extensible 3D Graphics for Web Authors”, Morgan Kaufmann, 2007
[3] Cass Everitt, Projective Texture mapping, 1999
[4] J.R. Spann and K.S. Kaufman, Photogrammetry using 3D Graphics and Projective Textures, IAPRS, 2000
[5] Eunjung Kim, Kwan-Hee Yoo, Je-Hoon Lee, Yong-Dae Kim, and Younggap You, Composite Endoscope Images from Massive Inner Interestine Photos, Lecture Notes on Artificial Intelligence 4570, pp.1042-1051, 2007 .
[6] C. Everitt, A. Rege, and C. Cebenoyan, Hardware Shadow Mapping. http://developer.nvidia.com/object/hwshadowmap_paper.html, 2001
[7] C. Everitt, Shadow Mapping. http://developer.nvidia.com/object/shadow_mapping.html. 2001
[8] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, Paul Haeberli “Fast shadows and lighting effects using texture mapping.” 19th annual conference on Computer Graphics and Interactive Techniques, pp.249-252, July 1992.
[9] Michalis Kamburelis, “Shadow maps and projective texturing in X3D”, the 15th International Conference on Web 3D Technology, p.17-26, 2010.
[10] In-Kwon Kim, Ho-Wook Jang, Jong-Sung Ha, Kwan-Hee Yoo, Specification and Implementation of Projective Texturing Node in X3D, International Journal of Contents, Vol.12, No.2, pp.1-5, June, 2016
[11] FreeWRL, “X3D Viewer”, http://freewrl.sourceforge.net/, June 2016.
[12] X3Dom, “X3D Viewer, http://www.x3dom.org. July, 2016

Appendix A. The Parser written in PERL language

A.1. TextureProjectorPerspective node

```
TextureProjectorPerspective => new
VRML::NodeType("TextureProjectorPerspective ",{
                    metadata => [SFNode, NULL, inputOutput,
"(SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 | SPEC_X3D33)"],
                    description => [SFString, "", inputOutput,
"(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 |
SPEC_X3D33)"],
                    location=> [SFVec3f, [0, 0, 1], inputOutput,
"(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 |
SPEC_X3D33)"],
                    direction => [SFVec3f, [0, 0, 1], inputOutput,
"(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 |
SPEC_X3D33)"],
                    upVector => [SFVec3f, [0, 1, 0], inputOutput,
"(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 |
SPEC_X3D33)"],
                    fieldOfView => [SFFloat, 45, inputOutput,
"(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 |
SPEC_X3D33)"],
                    aspectRatio => [SFFloat, 1, inputOutput,
"(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 |
SPEC_X3D33)"],
                    nearDistance => [SFFloat, 1, inputOutput,
"(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 |
SPEC_X3D33)"],
                    farDistance => [SFFloat, 10, inputOutput,
"(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 |
SPEC_X3D33)"],
                    global => [SFBool, FALSE, inputOutput,
"(SPEC_X3D31 | SPEC_X3D32 | SPEC_X3D33)"],
                    on => [SFBool, FALSE, inputOutput,
```

```
''(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 |
SPEC_X3D33)''],
                    texture=>[SFNode,NULL,inputOutput,
''(SPEC_VRML | SPEC_X3D30 | SPEC_X3D31 | SPEC_X3D32 |
SPEC_X3D33)''],
                    _dir =>[SFVec4f,[0,0,0,0],initializeOnly,0],
                    _loc =>[SFVec4f,[0,0,0,0],initializeOnly,0],
                    _upVec =>[SFVec4f,[0,0,0,0],initializeOnly,0],
                    __projTexture=>[SFNode,NULL,inputOutput, 0],
            }, ''X3DTextureProjectorNode''),
```

A.2. TextureProjectorParallel node

```
TextureProjectorParallel=>new VRML::NodeType(''TextureProjectorParallel'',{
                    metadata=>[SFNode, NULL, inputOutput, ''(SPEC_X3D30
|SPEC_X3D31|SPEC_X3D32|SPEC_X3D33)''],
                    description=>[SFString, '''', inputOutput, ''(SPEC_VRML|
SPEC_X3D30|SPEC_X3D31|SPEC_X3D32|SPEC_X3D33)''],
                    location=>[SFVec3f,[0,0,1],inputOutput, ''(SPEC_VRML|
SPEC_X3D30|SPEC_X3D31|SPEC_X3D32|SPEC_X3D33)''],
                    direction=>[SFVec3f,[0,0,1],inputOutput,''(SPEC_VRML
|SPEC_X3D30|SPEC_X3D31|SPEC_X3D32|SPEC_X3D33)''],
                    upVector=>[SFVec3f,[0,1,0],inputOutput,
''(SPEC_VRML|SPEC_X3D30|SPEC_X3D31|SPEC_X3D32|SPEC_X3D33)''],
                    fieldOfView=>[SFColorRGBA,[-1,1,-1,1],inputOutput,
''(SPEC_VRML|SPEC_X3D30|SPEC_X3D31|SPEC_X3D32|SPEC_X3D33)''],
                    aspectRatio=>[SFFloat,1,inputOutput,''(SPEC_VRML|
SPEC_X3D30|SPEC_X3D31|SPEC_X3D32|SPEC_X3D33)''],
                    nearDistance=>[SFFloat,1,inputOutput,''(SPEC_VRML|
SPEC_X3D30|SPEC_X3D31|SPEC_X3D32|SPEC_X3D33)''],
                    farDistance=>[SFFloat,10,inputOutput,''(SPEC_VRML|
SPEC_X3D30|SPEC_X3D31|SPEC_X3D32|SPEC_X3D33)''],
                    global=>[SFBool,FALSE,inputOutput,''(SPEC_X3D31|
SPEC_X3D32|SPEC_X3D33)''],
                    on=>[SFBool,FALSE,inputOutput,''(SPEC_VRML|
SPEC_X3D30|SPEC_X3D31|SPEC_X3D32|SPEC_X3D33)''],
                    texture=>[SFNode,NULL,inputOutput,''(SPEC_VRML|
SPEC_X3D30|SPEC_X3D31|SPEC_X3D32|SPEC_X3D33)''],
                    _dir=>[SFVec4f,[0,0,0,0],initializeOnly,0],
                    _loc=>[SFVec4f,[0,0,0,0],initializeOnly,0],
                    _upVec=>[SFVec4f,[0,0,0,0],initializeOnly,0],
                    __projTexture=>[SFNode,NULL,inputOutput,0],
            },''X3DGroupingNode''),
```