# Light Field Mapping:A Method for Progressive Encoding, Compression and Interactive Visualization of Surface Light Fields

Alexey Smirnov, Sergey Molinov, Dmitry Simakov, Radek Grzeszczuk
Intel Corporation

## 1 OVERVIEW

In this paper we present a new method for progressive encoding, efficient compression and interactive rendering of surface light fields called Light Field Mapping (LFM). The paper is organized as follows. In the first section there is an overview of LFM that describes all steps that build up this method. The second section describes implementation of LFM in VRML language. MPEG4 AFX committee accepted this implementation and now LFM is a part of MPEG4 AFX draft. Then in Appendix some technical details concerning LFM implementation are explained. The list of references concludes the paper.

A surface light field is a 4-dimensional function $f(r,s,\theta,\phi)$ that completely defines the radiance of every point on the surface of an object in every viewing direction. The first pair of parameters of this function $(r,s)$ describes the surface location and the second pair of parameters $(\theta,\phi)$ describes the viewing direction.

Intuitively, the LFM representation can be thought of as a special type of texture map that changes its appearance with the viewing angle. Because it is compact and allows for hardware-accelerated rendering, this representation is ideal for accurate modeling of the appearance of many physical and synthetic objects with complex surface reflectance properties. The system consists of three main components: approximation, compression and rendering that are described briefly.

### Approximation Step

The first component approximates the 4-dimensional surface light field function $f$ as a sum of a small number of products of lower-dimensional functions

$$f(r,s,\theta,\varphi) \approx \sum_{k=1}^{K} g_k(r,s)\, h_k(\theta,\phi)$$

**Equation 1:** Describes the approximation of light field data as a sum of a small number of produces of lower-dimensional functions.

The discrete functions $g_k$ and $h_k$ encode the light field data and are stored in a sampled form as texture maps, called *light field maps*. The functions $g_k$ are called the *surface maps* and functions $h_k$ as the *view maps* based on their parameterization. By taking advantage of the existing hardware support for texture mapping and composition surface light fields are visualized directly from the proposed representation at highly interactive frame rates. The process of rendering from this representation is called *light field mapping*.

### Compression Step

Although it is possible to render surface light fields directly from the approximation described in the last section, further compression of the surface light field data is possible. Since the light field maps are in essence collections of images that themselves exhibit redundancy, they can be further compressed using standard image compression techniques.

### Rendering Step

The sequence of rendering operations for each surface primitive is always the same. It starts with computing the view-dependent texture coordinates for the view maps. Subsequently, it proceeds to evaluate each approximation term and adds them together. Each approximation term is evaluated the same way:

1. The algorithm texture maps the surface primitive using the surface map.
2. It texture maps the surface primitive using the fragment of the view map determined by the view-dependent texture coordinates.
3. It performs pixel-by-pixel multiplication of the results of the two texture mapping operations. The following sections describe the rendering algorithm in more details.

### Partitioning of Data

Since the geometry of the models is represented as a triangular mesh, an obvious partitioning of the light field function is to split it between individual triangles. Unfortunately, an approximation of surface light field partitioned this way results in visible discontinuities at the edges of the triangles. To eliminate the discontinuities across triangle boundaries the light field data is partitioned around every vertex. The surface light field unit corresponding to each vertex is called the *vertex light field*. For vertex $v_j$, it is denoted as $f^{v_j}[r_p,q_p,\theta_q,\phi_q]$. Partitioning is computed by weighting the surface light field function

$$f^{v_j}[r_p, q_p, \theta_q, \phi_q] = \Lambda^{v_j}[r_p, q_p] f[r_p, q_p, \theta_q, \phi_q]$$

where $\Lambda^{v_j}[r_p, q_p]$ is the barycentric weight of each point in the ring of triangles around vertex $v_j$.

In the final step of vertex-centered partitioning each vertex light field is reparameterized to the local coordinate system of the vertex. To this end, the viewing direction angles are defined as the azimuth and elevation angles of the viewing direction vector in the local reference frame of the vertex. The vertex reference frame is defined in such a way that its z-axis is parallel to the surface normal at the vertex. The same letters are used to denote the local parameters of the vertex light field in order to simplify the notation.

Partitioning of light field data into vertex light fields ensures that in the resulting approximation each triangle shares its view maps with the neighboring triangles. Therefore, even though we approximate each vertex light field independently, we obtain an approximation that is continuous across triangles regardless of the number of approximation terms $K$ in Equation 1. Let $g^{v_j}[r_p, q_p]$ be the surface map and $h^{v_j}[\theta_q, \phi_q]$ be the view map corresponding to one approximation term of vertex light field $f^{v_j}[r_p, q_p, \theta_q, \phi_q]$, i.e.,

$$f^{v_j}[r_p, q_p, \theta_q, \phi_q] = g^{v_j}[r_p, q_p]\, h^{v_j}[\theta_q, \phi_q]$$

**Equation 2:** One term approximation of light field for vertex $v_j$.

Let $f^{\Delta_i}[r_p, q_p, \theta_q, \phi_q]$ denote the corresponding approximation term of the light field data for triangle $\Delta_i$. The following equality holds

$$f^{\Delta_i}[r_p, q_p, \theta_q, \phi_q] = \sum_{j=1}^{3} g_{\Delta_i}^{v_j}[r_p, q_p]\, h^{v_j}[\theta_q, \phi_q]$$

**Equation 3:** One term approximation of light field for triangle $\Delta_i$ expressed as a sum of the triangle's 3 vertex light fields.

In above equation index $j$ runs over the three vertices of triangle $\Delta_i$ and $g_{\Delta_i}^{v_j}[r_p, q_p]$ denotes the portion of the surface map corresponding to the triangle $\Delta_i$. This equality holds for all approximation terms.

## Rendering Algorithm

The rendering algorithm takes advantage of the property of vertex-centered partitioning, described by Equation 3, which says that the light field for each triangle can be expressed independently as a sum of its 3 vertex light fields. It enables a very efficient rendering routine that repeats the same sequence of operations for each mesh triangle. As each light field approximation term is evaluated the same way, the description of how to evaluate one approximation term of one triangle is sufficient to completely describe the rendering algorithm.

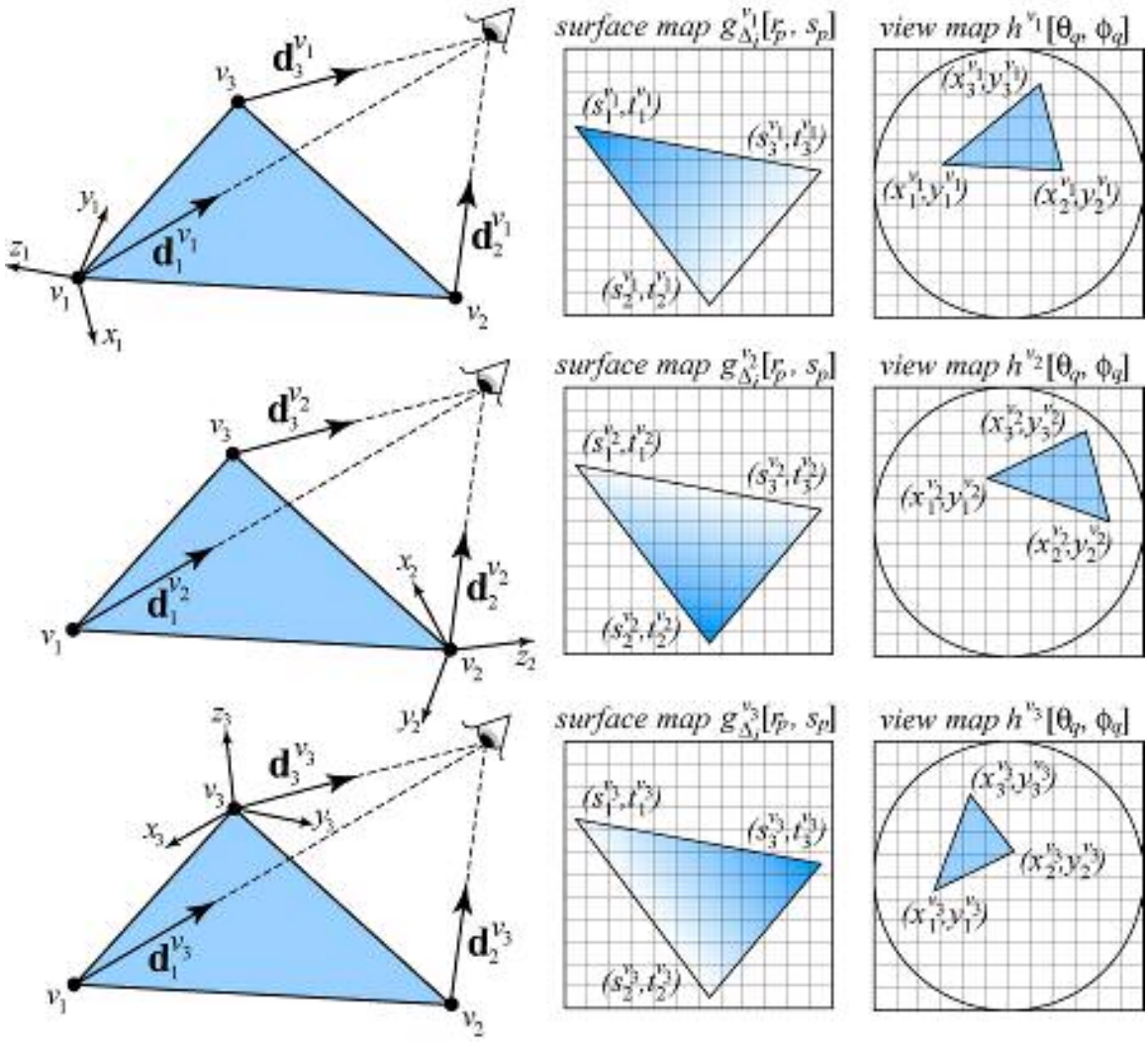


**Figure 1:** Light field maps for one approximation term of one triangle. Vertex reference frames are shown in the left column.

Figure 1 shows 6 light field maps used in Equation 3 to compute one approximation term of light field for triangle $\Delta_i$. The middle column shows surface maps $g_{\Delta_i}^{v_j}[r_p, q_p]$. In each image, the pixels covered by the shaded triangle correspond to the points inside triangle $\Delta_i$ where we sampled the light field function. We will describe the texture coordinates of these points as (*s*, *t*). As a result of the weighting applied during the construction of function $f^{v_j}[r_p, q_p, \theta_q, \phi_q]$ the pixels of the surface maps are weighted, as indicated in the figure by gradient shading, but this does not alter the rendering algorithm in any way.

The right column shows view maps $h^{v_j}[\theta_q, \phi_q]$. In each image, the pixels inside the circle correspond to the orthographic projection of the hemisphere of viewing directions, expressed in the local coordinate system *xyz* of vertex $v_j$, onto the plane *xy* shifted and scaled to the range (0,1). We will describe the texture coordinates of these points as (*x*,*y*). This projection allows a simple texture coordinate computation

$$x = (\mathbf{d} \cdot \mathbf{x} + 1)/2 \quad y = (\mathbf{d} \cdot \mathbf{y} + 1)/2$$

**Equation 4:** Equations describing parameterization of viewing directions.

In the equation above **d** represents the normalized local viewing direction and vectors **x** and **y** correspond to the axes of the local reference frame. Other transformations from 3D directions to 2D maps are possible but the one described here is efficient and accurate.

Based on where the camera is located, the rendering algorithm needs to access a different 2D subset of the 4D light field function. This is done by recomputing the view map coordinates $(x_i^{v_j}, y_i^{v_j})$ every time the camera moves. To this end, we apply Equation 4 to vectors $\mathbf{d}_i^{v_j}$ that represent the viewing directions to vertex $v_i$ expressed in the reference frame of vertex $v_j$. This results in 3 texture fragments shown in the right column of Figure 1. Note that the texture coordinates are different for each fragment because we are using a different reference frame to compute them. Note also that the surface map texture coordinates $(s_i^{v_j}, t_i^{v_j})$ do not depend on the viewing angle and they do not need to be recomputed when the camera moves.

Evaluating one complete approximation term is equivalent to multiplying pixel-by-pixel the image projections of the surface map and view map texture fragment pairs for the 3 vertex light fields of the triangle, each shown in a separate row of Figure 1 and adding the results together. The multiple term approximation of each triangle light field is computed by simply adding the results of each approximation term.

## Tiling of Light Field Maps

To avoid excessive texture swapping and improve rendering efficiency, we tile individual light field maps together to create tiled texture maps. Through out this document we are going to use term **tile** to refer to the image containing tiled light field maps. To simplify the problem, we allow a predefined set of triangle sizes during the resampling process, and then tile same-size light field maps together, as shown in Figure 2. We will use the term **surface map tile** to refer to the image containing tiled surface maps and we will use the term **view map tile** to refer to the image containing tiled view maps.

Since one triangle requires three surface maps per approximation term, all these maps are arranged in the same texture. The geometry of the model is split into $p$ segments so that the tiled view maps for each segment do not exceed maximum texture size allowed. One view map tile is produced per segment: [$V_1$, $V_2$, ..., $V_p$]. Let $[S_1^i, S_2^i, \ldots, S_{q_i}^i]$ be the list of surface map tiles for vertex tile $V_i$. Note that, in general, each triangle represented by a given tile will have more than one surface map in this tile. For each approximation term, the rendering algorithm is as follows

**for** $i$=1,...,$p$ **do**
  load view map tile into texture unit 1
  **for** $j$=1,...,$q_i$ **do**
    load surface map tile $S_j^i$ into texture unit 2
    render all triangles with surface maps in tile $S_j^i$
  **end for**
**end for**

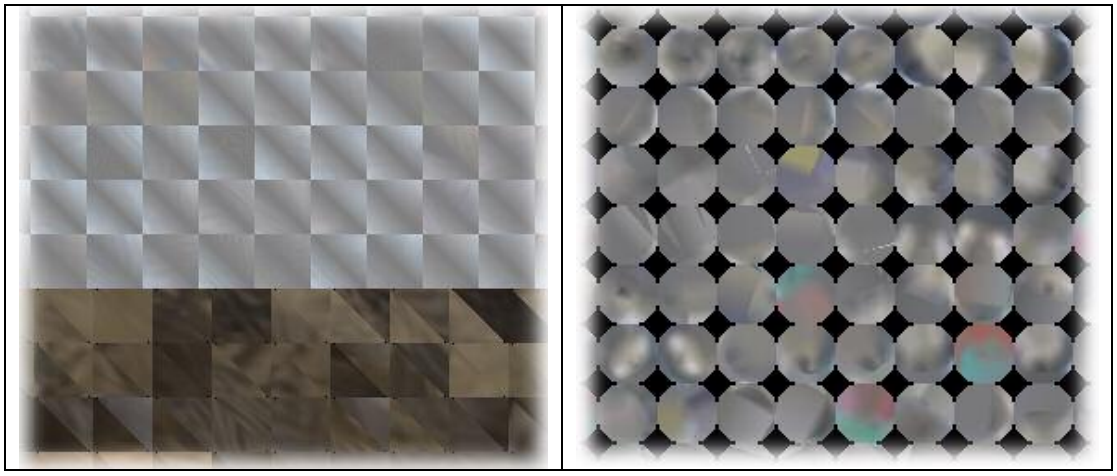

**Figure 2:** The tiled surface maps (left) and view maps (right)

## 2 NODE SPECIFICATION

In this section we describe implementation of LFM in VRML language. VRML provides a convenient way of describing complex 3D scenes. The basic element of this language is a node. VRML has a number of predefined nodes. The easiest way to describe content of a new type is to define new (custom) nodes.

The root node that describes the Light Field is LFM_Appearance. This node should be present in the appearance field of a Shape node, while its geometry field should contain an IndexedFaceSet.

```
LFM_Appearance
{
   exposedField   MFNode   tileList        []
   exposedField   MFNode   lightMapList    []
         exposedField     SFNode
         blendList                NULL
         exposedField     LFM_FrameList
         vertexFrameList          NULL
}
```

The field **tileList** describes the list of images containing the tiled light maps. Elements of this list should be nodes of type **ImageTexture**. In general, surface maps will be tiled separately from view maps and that not all of them are necessarily stored in a single image.

The LFM algorithm requires that each vertex of the mesh has a reference frame associated with it. The reference frames can be either computed automatically, as described in APPENDIX, or they can be specified explicitly through **LFM_FrameList** node. For vertices that do not have the reference frame specified through the **LFM_FrameList** node, the reference frames are computed using the automatic rules. A node describing the reference frames for the vertices is defined as

```
LFM_FrameList
{
  exposedField  MFInt32   index   [ -1 ]
  exposedField  MFVec3f   frame   [ 1 0 0, 0 1 0, 0 0 1 ]
}
```

The field **lightMapList,** used in the definition of the **LFM_Appearance** node, describes how to access individual light maps from within the image tiles. The number of the elements in the light map list corresponds to the number of the decomposition terms used in the surface light field approximation. The list **lightMapList** should contain elements of type **LFM_LightMap**. The format for accessing surface maps is slightly different from the format for accessing the view maps; therefore, each element in the light map list consists of a field describing how to access the surface maps and a field describing how to access the view maps. Note that each one of those two fields is going to be a list as well.

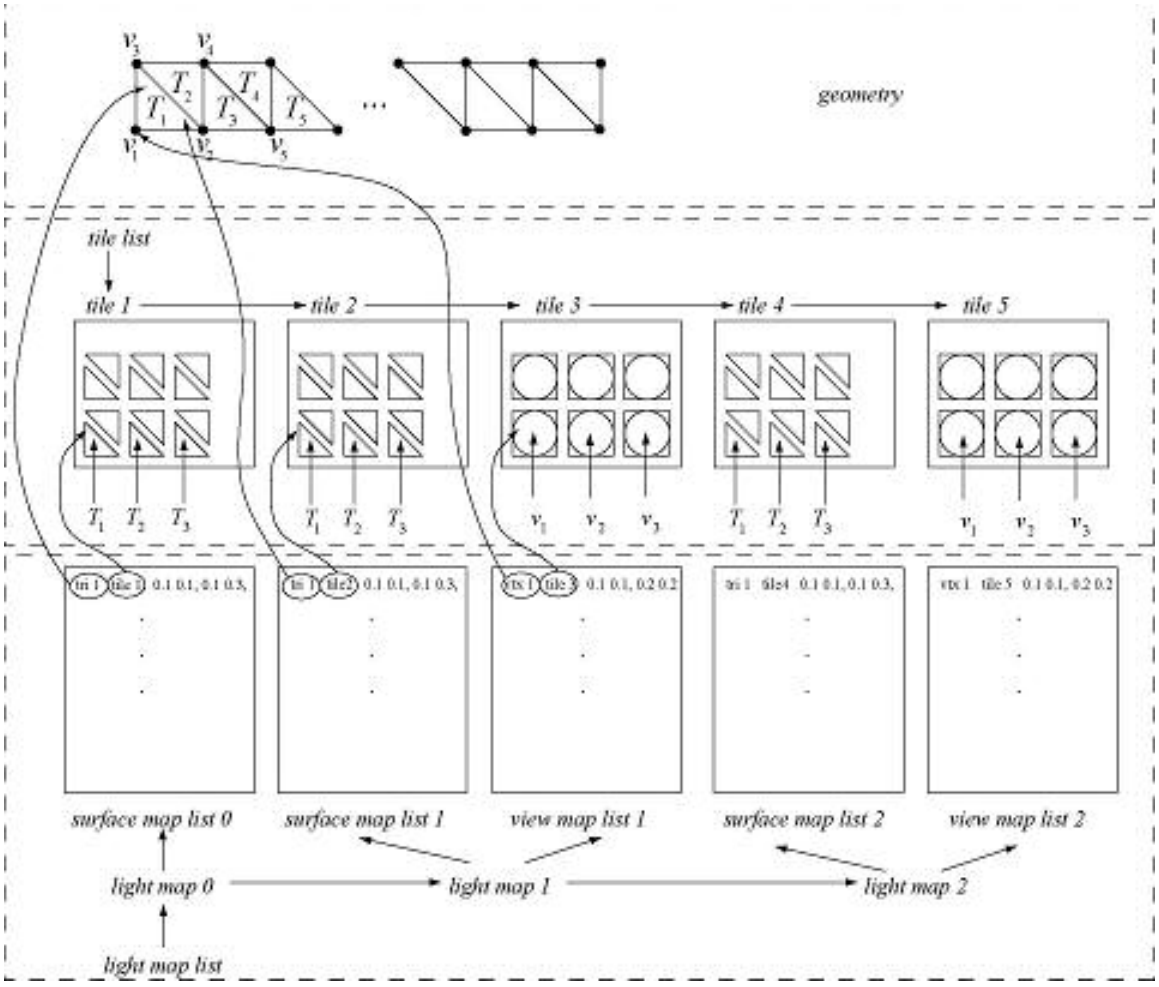


**Figure 3 -** This figure shows a diagram of the nodes used to define the shape that uses light field mapping appearance. The top of the figure shows the geometry of the object defined as a triangular. The middle of the figure shows the list of images containing the light map tiles. Finally, the bottom of the figure shows the light map list that specifies how to access individual light maps from within the tiles. Note that the first element of the light map list contains only a surface map list and no view map list. This element of the light map list would be used just like a regular texture map. The remaining two elements of the light map list each contain the par of surface map and view map list. In this case we would perform a multiplication of the corresponding surface maps and view maps. We made certain simplification in this diagram. For example, the diagram shows a one-to-one correspondence between the tiles and the surface/view map lists. In practice, this is not always the case, since we can have multiple lists for each tile.

**LFM_LightMap**
**{**
**exposedField SFNode surfaceMapList NULL**
**exposedField SFNode viewMapList NULL**
**exposedField SFVec3f scaleRGB 1 1 1**
**exposedField SFVec3f biasRGB 0 0 0**
**exposedField SFInt32 priorityLevel 0**
**}**

**Scale** and **bias** map the default color range [0, 1] of the image into the color range required for proper image synthesis. Let **s** be the RGB scale vector, let **b** be the RGB bias vector and let **c** be the RGB color value. The new color range $\mathbf{c}_{new}$ for color vector **c** can be computed using vectors **s** and **b** as follows

$$\mathbf{c}_{new} = \mathbf{s}\,\mathbf{c} + \mathbf{b}$$

**Equation 5 – Scale and Bias map default color range to image synthesis color range..**

The field **priorityLevel** is a non-negative integer value specifying the level of importance of a given **LightMapList** node. The lower the value associated with the node, the more important it is. If the value is **0**, the node must be rendered. When the renderer is set to a given priority value, then all nodes with the **priorityLevel**

below that value must be rendered. For example, if the priority value of the render is set to 2, then all nodes with priority level 0, 1, and 2 must be rendered.

The lists **surfaceMapList** and **viewMapList** containing information about the surface maps and the view maps are defined as follows

**LFM_SurfaceMapList**
**{**
**exposedField MFInt32 triangleIndex []**
**exposedField MFInt32 tileIndex []**
**exposedField MFInt32 viewMapIndex []**
**exposedField SFNode triangleCoordinate NULL**
**}**

where **triangleIndex** refers to the index of a given triangle as defined by the geometry node, **tileIndex** refers to the index of the tile as defined by the list of tiles, **viewMapIndex** indicates the number of view maps that should be used to render a given triangle (it equals –1 if there is no **viewMapList** in the current lightMap) and **triangleCoord** is a list of triples of texture coordinates of every triangle that specifies which part of the tile contains the surface map for the current triangle.

**LFM_ViewMapList**
**{**
**exposedField MFInt32 vertexIndex []**
**exposedField MFInt32 tileIndex []**
**exposedField SFNode textureOrigin NULL**
**exposedField SFNode textureSize NULL**
**}**

where **vertexIndex** refers to the list of indices of vertices as defined by the geometry node, **tileIndex** refers to the list of indices of the tiles as defined by the list of tiles. The fields **textureOrigin** and **textureSize** are two lists of texture coordinates that specify the origin and the size of the view map, respectively, they should contain **TextureCoordinate nodes**.

The field **blendList,** used in the definition of the **LFM_Appearance** node, describes how to combine the individual light maps together during rendering. It may contain an **LFM_blendList** node

**LFM_blendList**
**{**
**exposedField MFInt32 lightMapIndex**
**exposedField MFInt32 blendMode**
**}**

where **lightMapIndex** refers to the index of the lightMap in **lightMapList** and **tileIndex** refers to the index of the tile as defined by the list of tiles and the field **blendMode** specifies what type of blending operation to use when combining the given **LightMapList** node with the data in the framebuffer. This field can take on the following values: 0 or LFM_ADD, 1 or LFM_SUBTRACT.

Note the following implementation facts

1. For each triangle, each approximation term consists of 3 surface map/view map pairs.
2. In general, each element in the **LFM_LightMap** will have a pair of valid pointers, one pointing to a valid **LFM_SurfaceMapList** node and one pointing to a valid **LFM_ViewMapList** node. If this is the case, during rendering the corresponding surface maps and view maps get multiplied together and blended with the earlier rendering results.
3. In case, when an element of the **LFM_LightMap** list contains a valid pointer to the **LFM_SurfaceMapList** node, but the pointer to the **LFM_ViewMapList** node is **NULL**, we simply use the surface maps as ordinary texture maps.
4. Let **sm1**, **vm1**, ..., **smK**, **vmK** be the first K pairs of surface map/view map for a given triangle. Multitexturing can be supported in the form **sm0 + sm1*vm1 + sm2*vm2 + sm3*vm3 + ...**, where addition refers to adding (or subtracting, depending on the value of field **blendMode**) the results of rendering through blending, and multiplication refers to pixel-by-pixel modulation of rendering results. This set of operations is repeated for each mesh triangle of the object.

## 3 APPENDIX

The rule for computing the reference frame of a given triangle consists of two steps:

1. Compute the following vectors

$$\mathbf{e}_1^{\Delta_i} = \mathbf{v}_1 - \mathbf{v}_2,\ \mathbf{e}_2^{\Delta_i} = \mathbf{v}_2 - \mathbf{v}_3,\ \mathbf{e}_3^{\Delta_i} = \mathbf{e}_1^{\Delta_i} \times \mathbf{e}_2^{\Delta_i}$$

where $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ are the vectors describing the positions of the 3 vertices of the given triangle.

2. Construct the change of basis matrix **R** = [ **x y z** ], where

$$\mathbf{x} = \frac{\mathbf{e}_1^{\Delta_i}}{\left\|\mathbf{e}_1^{\Delta_i}\right\|},\ \mathbf{z} = \frac{\mathbf{e}_3^{\Delta_i}}{\left\|\mathbf{e}_3^{\Delta_i}\right\|},\ \mathbf{y} = \mathbf{x} \times \mathbf{z}.$$

The rule for computing the reference frame of a vertex consists of the following steps:

1. Compute the direction of the normal of the vertex to be the average of the directions of the ring of triangles for that vertex
   where $R_j$ is the set of triangles contained in the ring around vertex $v_j$.

$$\mathbf{e}_3^{v_j} = \sum_{i \in R_j} \mathbf{e}_1^{\Delta_i} \times \mathbf{e}_2^{\Delta_i}$$

2. Pick the first edge of the first triangle in the ring as the direction of the x-axis of the vertex reference frame

$$\mathbf{e}_1^{v_j} = \mathbf{e}_1^{\Delta_1}$$

3. Construct the change of basis matrix **R** = [ **x y z** ], where

$$\mathbf{x}' = \frac{\mathbf{e}_1^{v_j}}{\left\|\mathbf{e}_1^{v_j}\right\|},\ \mathbf{z} = \frac{\mathbf{e}_3^{v_j}}{\left\|\mathbf{e}_3^{v_j}\right\|},\ \mathbf{y} = \mathbf{x}' \times \mathbf{z},\ \mathbf{x} = \mathbf{z} \times \mathbf{y}.$$

## 4 REFERENCES


1. W-C. Chen, R. Grzeszczuk, J-Y. Bouguet. Light Field Mapping: Hardware-Accelerated Visualization of Surface Light Fields. Part of "Acquisition and Visualization of Surface Light Fields," *SIGGRAPH 2001 Course Notes for Course #46*. Available from http://www.intel.com/research/mrl/research/lfm



**Authors:**
Alexey Smirnov, Sergey Molinov, Dmitry Simakov, Radek Grzeszczuk
Intel Corporation

e-mail: {alexeyx.smirnov, sergey.molinov, dmitryx.simakov, radek.grzeszczuk}@intel.com