Extensible 3D (X3D)
Part 1: Architecture and base components

12 Shape component

12.1 Introduction

12.1.1 Name

The name of this component is "Shape". This name shall be used when referring to this component in the COMPONENT statement (see 7.2.5.4 Component statement).

12.1.2 Overview

This clause describes the Shape component of this part of ISO/IEC 19775. The Shape component defines nodes for associating geometry with their visible properties and the scene environment. Table 12.1 provides links to the major topics in this clause.

Table 12.1 — Topics

12.2 Concepts

12.2.1 Shape characteristics

The Shape node associates a geometry node with nodes that define that geometry's appearance. Shape nodes shall be part of the transformation hierarchy to have any visible result, and the transformation hierarchy shall contain Shape nodes for any geometry to be visible (the only nodes that render visible results are Shape nodes and the background nodes described in 24 Environmental effects). A Shape node contains exactly one geometry node in its *geometry* field, which is of type *X3DGeometryNode*. The Shape node descends from the abstract base type *X3DShapeNode*.

12.2.2 Appearance characteristics

Shape nodes may specify an Appearance node that describes the appearance properties (material, texture and texture transformation) to be applied to the ~~Shape's geometry~~ geometry of the Shape. All valid children of the Appearance node descend from the abstract base type *X3DAppearanceChildNode*.

Nodes of the following types may be specified in the *material* field of the Appearance node:

- Material
- PhysicalMaterial
- TwoSidedMaterial (deprecated)
- UnlitMaterial

This set of nodes may be extended by creating new nodes derived from the *X3DMaterialNode* abstract base type.

Nodes of the following types may be specified in the *backMaterial* field of the Appearance node:

- Material
- PhysicalMaterial
- UnlitMaterial

This set may be extended by creating new nodes derived from the *X3DOneSidedMaterialNode* abstract base type.

The Appearance node specifies texture mapping in its *texture* field. Valid values of the texture field are descendants of *X3DTextureNode*, including:

- ImageTexture
- PixelTexture
- MovieTexture

- MultiTexture

This set may be extended by creating new nodes derived from the abstract *X3DTextureNode* base class as defined in 18.3.2 X3DTextureNode.

Nodes of the type *X3DTextureTransformNode* may be specified in the *textureTransform* field of the Appearance node (see 18.3.4 X3DTextureTransformNode), including:

- TextureTransform

Interaction between the appearance properties and properties specific to geometry nodes are described in 13 Geometry3D component and 14 Geometry2D component.

An Appearance node may specify additional information about the appearance of the corresponding geometry. The *acousticProperties* field can be provided by an AcousticProperties node. Special properties may be defined for lines and filled areas. These properties are defined in the *lineProperties*, *fillProperties* and *pointProperties* fields by the following nodes, respectively:

- LineProperties
- FillProperties
- PointProperties

12.2.3 Two-sided materials

A polygon defines a front face based on the direction of the normal. That normal is either explicitly provided by the end user or implicitly calculated by the browser based on the winding rules for the node (for example, see the *ccw* field on many of the polygonal geometry nodes such as IndexedFaceSet).

~~The TwoSidedMaterial node provides a way to render the front and back sides of the polygon with different material properties. Figure 12.1 depicts the effects of the TwoSidedMaterial node.~~

The *solid* field, described in the 11.2.3 Common geometry fields, controls whether the geometry is visible from the back side.

- When *solid* is TRUE, the back faces are not visible at all.
- When *solid* is FALSE, the back faces of the geometry are lit, using an inverted normal vector than the corresponding front faces.

Using the Appearance field *backMaterial* allows rendering the front and back sides of the polygon with different material properties. It is meaningful only when *solid* is FALSE, since otherwise the back faces are never rendered. When the *solid* is TRUE, the value of *backMaterial* has no effect on rendering.

TwoSidedMaterial (deprecated) provides similar functionality (through the various *back...* fields) but is limited to the Phong lighting model.

Figure 12.1 depicts example effects of the Appearance field *backMaterial* node.

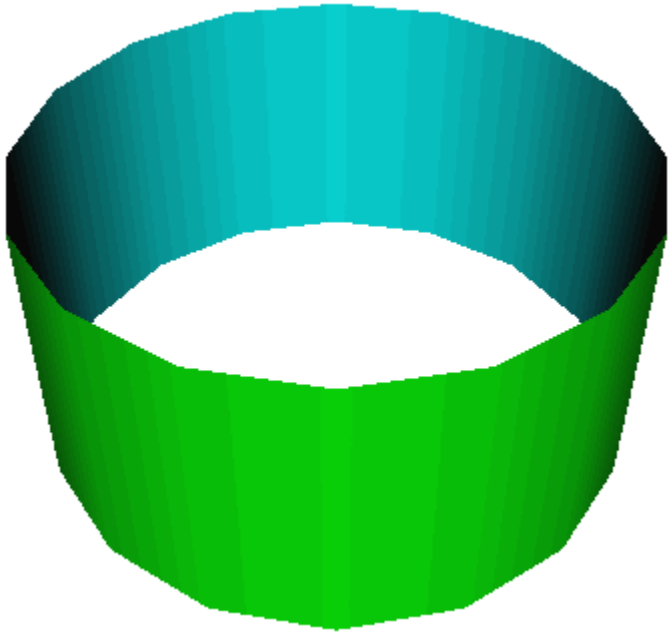

Figure 12.1 — Effects of different material properties on front and back sides of the geometry

Several constraints pertain to the *backMaterial* field value: ~~, to make the definition reasonable and easy to implement by the browsers.~~

- The *backMaterial* field can have a value different than NULL only when the *material* field also has a value different than NULL.
- It is not allowed to provide a *backMaterial* when the *material* specifies a TwoSidedMaterial (deprecated).
- When both the *material* and *backMaterial* are provided (not NULL), it is required that they:
    1. Specify the same node class. In other words, both of them should be Material, or both should be PhysicalMaterial, or both should be UnlitMaterial.

2. Use the same textures with the same mapping. In other words, the values of the fields *xxxTexture* and *xxxTextureMapping* documented in 12.2.4 Texture mapping specified in material nodes shall be equal for both front and back materials. The author should use the DEF / USE mechanism to have the same references to texture nodes.

In effect, the front and back material parameters may differ only in their scalar or vector values. For example, front side may have a different *diffuseColor* than the back side.

To summarize, the following combinations are allowed:

- Both *material* and *backMaterial* are NULL. In this case we have an unlit (pure white) material, when viewed from both the front and back side.

  This case is *exactly equivalent* to using an UnlitMaterial with default *emissiveColor* white for both *material* and *backMaterial*.

- *material* is a *TwoSidedMaterial* node, *backMaterial* is NULL.

  This case is only provided for compatibility. The *TwoSidedMaterial* is deprecated since X3D 4.0. Whether the rendering parameters are the same, or different, for front and back sides is determined by the *separateBackColor* field of the relevant *TwoSidedMaterial* node.

- *material* contains a Material, PhysicalMaterial or UnlitMaterial node, and *backMaterial* is NULL.

  In this case, both front and back sides will be rendered with the same parameters. This case is *exactly equivalent* to reusing the same material node (through DEF / USE mechanism) for both *material* and *backMaterial* fields.

- *material* contains a Material, PhysicalMaterial or UnlitMaterial node, and *backMaterial* also contains a node of the same type.

  In this case, some front and back material parameters may differ.

12.2.4 Texture mapping specified in material nodes

The X3DOneSidedMaterialNode and descendants (Material, PhysicalMaterial, UnlitMaterial) introduce a number of fields to modify material parameters using textures. They are consistently defined by a pair of fields like this:

```
SFNode   [in,out] xxxTexture        NULL
SFString [in,out] xxxTextureMapping ""
```

The field *xxxTexture* indicates a texture node.

The *xxxTextureMapping* determines the *texture coordinates* and *texture coordinates transformation* for given texture *xxxTexture*.

The corresponding texture coordinate and texture coordinate transformation nodes have a field *mapping* that will match the value of the *xxxTextureMapping* field. See the X3DSingleTextureCoordinateNode and X3DSingleTextureTransformNode definitions.

Multiple textures may use the same texture coordinates and their transformations. For example, it is common that both `normalTextureMapping` and `diffuseTextureMapping` are equal, if the graphic artist prepared both `normalTexture` and `diffuseTexture` simultaneously, assuming the same mapping.

12.2.4.1 Texture coordinates

Let's define a *list of texture coordinates* for each geometry node like this:

- If the geometry field doesn't have a *texCoord* field then this list is empty.

  Most geometric objects with a predefined geometry (e.g. *Sphere*) don't have *texCoord* field. Most geometric objects with geometry defined by the author (e.g. all the nodes derived from X3DComposedGeometryNode) have *texCoord* field.

- Otherwise, if the value of the *texCoord* field is `NULL`, then this list is again empty.
- Otherwise, if the value of the *texCoord* field is a single node derived from X3DSingleTextureCoordinateNode, then place this one node on the list.

  X3DSingleTextureCoordinateNode includes all texture coordinate nodes (like TextureCoordinate or TextureCoordinateGenerator) except MultiTextureCoordinate.

- Otherwise, the value of this field must be MultiTextureCoordinate node. Then use the *MultiTextureCoordinate.texCoord* contents list as our *list of texture coordinates*.

Note: The above definition means that using a MultiTextureCoordinate with exactly one child is equivalent to using this child directly. This is a general rule in X3D 4.0, see also the MultiTextureCoordinate specification for details and an example.

All the X3DSingleTextureCoordinateNode nodes on the *list of texture coordinates* defined above must have a different *mapping* value. An exception is the empty *mapping* value, which may occur many times.

If the *xxxTextureMapping* field is not empty, it must refer to a corresponding *X3DSingleTextureCoordinateNode* node on a *list of texture coordinates*.

If the *xxxTextureMapping* field is empty, then the **first** item on a *list of texture coordinates* is used (regardless of it's *mapping* value). Only if no such texture coordinate exists (the list is empty), then then *default texture coordinates* for the specific geometry node are used.

The algorithm to perform the *default texture coordinate calculation* is described at each geometry node. For example IndexedFaceSet determines the coordinates based on the local bounding box sizes, Box has the texture applied 6 times on 6 faces etc.

> Hint for implementations: This section makes an important guarantee. Generating *default texture coordinates* only needs to be done when the *texCoord* field of the geometry is empty, or contains an empty *MultiTextureCoordinate* node. In all other cases, you know that *default texture coordinates* are not necessary, because all textures will use one of the coordinates in the *texCoord* list.
>
> This is an important property, because browsers may want to avoid generating *default texture coordinates* as it is a time-consuming process (e.g. requires to iterate over vertexes at least twice in case of *IndexedFaceSet*) and often not necessary (models exported by 3D authoring software typically have all texture coordinates provided in the file).
>
> So we wanted to enable this optimization, and make it easy to detect looking only at *texCoord* contents. In effect, it doesn't matter what Appearance or material will be used with this geometry node — you can easily avoid most cases when *default texture coordinates* would be unused just by inspecting the geometry *texCoord* value.

12.2.4.2 Texture coordinates transformation

Let's define a *list of texture transformations* for each geometry node like this:

- If the shape uses no Appearance node then this list is empty.
- Otherwise, if the value of *Appearance.textureTransform* is NULL, then this list is again empty.
- Otherwise, if the value of *Appearance.textureTransform* is a single node X3DSingleTextureTransformNode, then place this one node in the list.

  Most texture transformation nodes are derived from X3DSingleTextureTransformNode, like TextureTransform and TextureTransform3D. But not MultiTextureTransform.

- Otherwise, the value of *Appearance.textureTransform* must be MultiTextureTransform. Then use the *MultiTextureTransform.textureTransform* contents as our *list of texture transformations*.

Note that we treat a MultiTextureTransform with a single child always the same as using this child directly. This is a general rule in X3D 4.0, see also the MultiTextureTransform specification for details and an example.

If the *xxxTextureMapping* field is not empty, it must refer to a corresponding *X3DSingleTextureTransformNode* node within the *list of texture transformations*. The *X3DSingleTextureTransformNode* node must have equal *mapping* value.

If the *xxxTextureMapping* is an empty string, then the **first** item on a *list of texture transformations* is used (regardless of it's *mapping* value). If the list is empty, no texture transformation is used.

Note: Throughout this section, we treat empty string as a special case for *xxxTextureMapping* and *mapping* fields. Such mapping names are allowed (they are even the default) but they do not constitute a "match". It would be error-prone if two empty mapping values would match, as it's easy to use them accidentally, since they are the default field values. Instead, empty *xxxTextureMapping* just indicates "use the first coordinates / transformations" — this is simplest and most natural.

12.2.5 Coexistence of textures specified in material nodes with the "Appearance.texture" field

In X3D 4.0, models can specify textures using the *xxxTexture* fields inside the various X3DOneSidedMaterialNode descendants. This allows to control every material parameter by a different texture.

Alternatively, models can also use the mechanism known from X3D 3.x, and provide a texture inside the *Appearance.texture* field. This is also the only way to use the MultiTexture node (which cannot be placed in *xxxTexture* fields, as it would make implementation complicated).

The exact behavior of *MultiTexture* node and *Appearance.texture* is this:

1. If the *Appearance.material* is Material, and the *Material.diffuseTexture* is `NULL`, then *Appearance.texture* affects the *diffuseParameter* for the lighting equation.

   In a way, *Appearance.texture* performs then the role of *Material.diffuseTexture*. It can even use *MultiTexture* to calculate the diffuse parameter by a composition (e.g. addition or multiplication) of other textures.

   The *Material.diffuseTextureMapping* value doesn't matter in this case.

2. If the *Appearance.material* is PhysicalMaterial, and the *PhysicalMaterial.baseTexture* is NULL, then *Appearance.texture* affects the *baseParameter* for the lighting equation.

   The *PhysicalMaterial.baseTextureMapping* value doesn't matter in this case.

3. If the *Appearance.material* is UnlitMaterial, and the *UnlitMaterial.emissiveTexture* is NULL, then *Appearance.texture* affects the *emissiveParameter* for the lighting equation.

   The *UnlitMaterial.emissiveTextureMapping* value doesn't matter in this case.

4. Otherwise, if the *Appearance.material* is NULL, then we behave as if the UnlitMaterial with all fields at default was used. So the *Appearance.texture* affects the *emissiveParameter* for the lighting equation, and is used with the unlit lighting model.

Note that when the *Appearance.texture* is used to calculate one of the parameters described above, the texture coordinates/transformations are determined following the MultiTexture specification. This means that MultiTextureCoordinate and MultiTextureTransform nodes can be used, with the order corresponding to the order of textures inside *MultiTexture.texture* list. If the *Appearance.texture* is not *MultiTexture* then the first set of texture coordinates/transformations are used. See the MultiTextureCoordinate and MultiTextureTransform specification for details.

The 17 Lighting component describes the exact equations to calculate the lighting parameters, consistent with the above description.

12.3 Abstract types

12.3.1 *X3DAppearanceChildNode*

```
X3DAppearanceChildNode : X3DNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This is the base node type for the child nodes of the *X3DAppearanceNode* type.

12.3.2 *X3DAppearanceNode*

```
X3DAppearanceNode : X3DNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This is the base node type for all Appearance nodes.

12.3.3 *X3DMaterialNode*

```
X3DMaterialNode : X3DAppearanceChildNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
}
```

This is the base node type for all material nodes.

There are two direct descendants of this node type:

1. Abstract X3DOneSidedMaterialNode.

   In turn, the *X3DOneSidedMaterialNode* is a descendant for all non-abstract and non-deprecated material nodes that you shall use in X3D models:

   - Material (Phong lighting model)
   - PhysicalMaterial (physically-based lighting model)
   - UnlitMaterial (trivial lighting model that ignores light sources, for non-realistic rendering and special effects)
2. TwoSidedMaterial (deprecated)

12.3.4 *X3DOneSidedMaterialNode*

```
X3DOneSidedMaterialNode : X3DMaterialNode {
  SFColor  [in,out] emissiveColor          0 0 0  [0, 1]
  SFNode   [in,out] emissiveTexture        NULL   [X3DSingleTextureNode]
  SFString [in,out] emissiveTextureMapping  ""

  SFNode   [in,out] metadata               NULL   [X3DMetadataObject]

  SFFloat  [in,out] normalScale            1      [0, ∞]
  SFNode   [in,out] normalTexture          NULL   [X3DSingleTextureNode]
  SFString [in,out] normalTextureMapping    ""
}
```

Editorial note: We consider merging this abstract node with *X3DMaterialNode*. One one hand, it would simplify the hierarchy. On the other hand, (deprecated) *TwoSidedMaterial* would be left in a weird state, with fields it doesn't use (like *emissiveTexture*, *normalTexture*...). Implementations that still support *TwoSidedMaterial* would need to "invent" a class similar to "X3DOneSidedMaterialNode" on their own. See here for details.

This is the base node type for material nodes that describe how the shape looks like from one side.

This node defines common properties for a lighting calculation, but independent of the lighting model (Phong, physically-based, unlit).

This node can be used within *Appearance.material* or *Appearance.backMaterial*.

The *normalTexture* field affects normal vectors information (surface curvature) in the following way:

- Each normal encoded in a texture is a 3D vector (normalized direction).

  3D direction of each normal shall be calculated from texture RGB color, using this equation:

  *normal.xyz = normalize((textureSample(normalTexture).rgb * vec3(2,2,2) - vec3(1,1,1)) * vec3(normalScale, normalScale, 1))*

  That is, assuming *normalScale* equal 1 (default), the red color component is linearly mapped from [0..1] to [-1..1] range and represents the X axis of the normal vector. Analogously the green component is mapped to Y, and the blue component is mapped to Z.

- The normals are provided in the *tangent space*.

  In tangent space:

  - The (0,0,1) vector is pointing perfectly outward from a polygon.

    More precisely, the "outward" direction (mapped to (0,0,1) in tangent space) is the direction of the "normal vector" derived from other X3D mechanisms: from the per-vertex or per-face normal vectors (if provided in the *Normal* node), or calculating the normals automatically (e.g. using *IndexedFaceSet.creaseAngle*).

  - The vectors (1,0,0) and (0,1,0) in the tangent space indicate the direction where the texture coordinate U and V grows. Naturally, they are adjusted to be always orthogonal to (0,0,1) and each other.

    In the future we may add to the X3D standard a way to provide explicit tangent vectors. In X3D 4.0, the implementation should always calculate tangent and bitangent vectors using a standard algorithm, like the *MikkTSpace algorithm*.

- Observe that a correct normalmap texture is typically blueish, since most of the normals on a more-or-less smooth surface revolve around (0,0,1), thus the texture colors revolve around (0.5,0.5,1).

- Alpha channel of the *normalTexture* is ignored by the calculations. X3D browsers *can* use the alpha channel of the *normalTexture* to specify heights (from which the normal vectors have been derived). In the current X3D standard version, these heights are not used for anything, although browsers may already use them for browser-specific rendering effects (for example to perform *parallax bump mapping* or *displacement*, activated by browser-specific extensions).

The *emissiveColor*, together with *emissiveTexture*, allow to model "glowing" objects. This can be useful for displaying unlit (pre-lit) models (where the light energy of the room is computed explicitly), or for displaying scientific data. To display an "unlit" object (whose visible color should not be modified by any light in the scene), author can use UnlitMaterial node.

The *emissiveTexture* RGB channel is multiplied with the *emissiveColor* to yield the *emissiveParameter* in the lighting equations.

The meaning of the alpha channel of the *emissiveTexture* depends on the *X3DOneSidedMaterialNode* descendant. It is ignored by Material and PhysicalMaterial. It is used, as the transparency factor, by the UnlitMaterial. Across the specification, the treatment of *Material.diffuseTexture*, *PhysicalMaterial.baseTexture* and *UnlitMaterial.emissiveTexture* is consistent: these "main" textures provide the transparency information for given material. For details, refer to the documentation of each *X3DOneSidedMaterialNode* descendant.

See the section 12.2.4 Texture mapping specified in material nodes for a description how the texture coordinates and texture coordinate transformations are determined based on the *xxxTextureMapping* fields of this node.

12.3.5 *X3DShapeNode*

```
X3DShapeNode : X3DChildNode, X3DBoundedObject {
  SFNode  [in,out] appearance  NULL     [X3DAppearanceNode]
  SFBool  [in out] bboxDisplay FALSE
  SFNode  [in,out] geometry    NULL     [X3DGeometryNode]
  SFNode  [in,out] metadata    NULL     [X3DMetadataObject]
  SFBool  [in out] visible     TRUE
  SFVec3f []       bboxCenter  0 0 0    (-∞,∞)
  SFVec3f []       bboxSize    -1 -1 -1 [0,∞) or -1 -1 -1
}
```

This is the base node type for all Shape nodes.

12.4 Node reference

12.4.1 AcousticProperties

```
AcousticProperties : X3DAppearanceChildNode  {
  SFFloat [in,out] absorption 0    [0,1]
  SFFloat [in,out] diffuse    0    [0,1]
  SFNode  [in,out] metadata   NULL [X3DMetadataObject]
  SFFloat [in,out] refraction 0    [0,1]
  SFFloat [in,out] specular   0    [0,1]
}
```

The AcousticProperties node specifies the interaction of sound waves with the characteristics of objects in the scene. Properties influencing sound propagation include surface-related physical phenomena such as the *specular* reflection, *diffuse* reflection, *absorption*, and *refraction* coefficients of materials. These coefficient values are expected to fully account for physical and structural characteristics of the associated geometry such as width, height, thickness, shape, softness and/or hardness, and density variations.

The *absorption* field specifies the sound absorption coefficient of a surface which is the ratio of the sound intensity absorbed or otherwise not reflected by a specific surface that of the initial sound intensity. This characteristic depends on the nature and thickness of the material. Sound energy is partially absorbed when it encounters fibrous or porous materials, panels that have some flexibility, volumes of air that resonate, and openings in room boundaries (e.g. doorways). Moreover, the absorption of sound by a particular shape depends on the angle of incidence and frequency of the sound wave.

The *diffuse* field describes the diffuse coefficient of sound reflection. This is one of the physical phenomena of sound that occurs when a sound wave strikes a plane surface, and part of the sound energy is reflected back into space in multiple directions.

The *refraction* field describes the sound refraction coefficient of a medium, which determines the change in propagation direction of a sound wave when it obliquely crosses the boundary between two mediums where its speed is different. These relationships are described by Snell's Law.

The *specular* field describes the specular coefficient of sound reflection, which is one of the physical phenomena of sound that occurs when a sound wave strikes a plane surface. Part of the sound energy is directly reflected back into space, where the angle of reflection is equal to the angle of incidence.

12.4.2 Appearance

```
Appearance : X3DAppearanceNode {
  SFNode [in,out] acousticProperties NULL [AcousticProperties]
  SFNode [in,out] backMaterial       NULL [X3DOneSidedMaterialNode]
  SFNode [in,out] fillProperties     NULL [FillProperties]
  SFNode [in,out] lineProperties     NULL [LineProperties]
  SFNode [in,out] material           NULL [X3DMaterialNode]
  SFNode [in,out] metadata           NULL [X3DMetadataObject]
  SFNode [in,out] pointProperties    NULL [PointProperties]
  MFNode [in,out] shaders            []   [X3DShaderNode]
```

```
    SFNode [in,out] texture           NULL [X3DTextureNode]
    SFNode [in,out] textureTransform  NULL [X3DTextureTransformNode]
}
```

The Appearance node specifies the visual properties of geometry. The value for each of the fields in this node may be NULL. However, if the field is non-NULL, it shall contain one node of the appropriate type.

The *acousticProperties* field, if specified, shall contain an AcousticProperties node describing coefficients related to the physical propagation of sound for various materials.

The *material* field, if specified, shall contain a Material, PhysicalMaterial, TwoSidedMaterial (deprecated) or UnlitMaterial node. If the *material* field is NULL or unspecified, lighting is off (all lights are ignored during rendering of the object that references this Appearance) and the unlit object colour is (1, 1, 1). Details of the X3D lighting model are in 17 Lighting component.

The *backMaterial* field, if specified, shall contain a Material, PhysicalMaterial or UnlitMaterial node. It is only allowed to define a *backMaterial* if the *material* is also defined (not NULL). The node type provided to *backMaterial* (if any) must match the node type provided to *material*. This field allows to render back faces with a different material parameters than the front faces. The meaning and all constraints of this field are explained in the section Two-sided materials.

The *texture* field, if specified, shall contain one of the various types of texture nodes (see 18 Texturing component). If the texture node is NULL or the *texture* field is unspecified, the object that references this Appearance is not textured.

The *textureTransform* field, if specified, shall contain a TextureTransform node as defined in 18.4.8 TextureTransform. If the *textureTransform* is NULL or unspecified, the *textureTransform* field has no effect.

The *fillProperties* field, if specified, shall contain a FillProperties node. If *fillProperties* is NULL or unspecified, the *fillProperties* field has no effect.

The *lineProperties* field, if specified, shall contain a LineProperties node. If *lineProperties* is NULL or unspecified, the *lineProperties* field has no effect.

The *pointProperties* field, if specified, shall contain a PointProperties node. If *pointProperties* is NULL or unspecified, the *pointProperties* field has no effect.

The *shaders* field contains a listing, in order of preference, of nodes that describe programmable shaders that replace the fixed rendering requirements of this part of ISO/IEC 19775 with user-provided functionality. If the field is not empty, one shader node is selected and the fixed rendering requirements defined by this

specification are ignored. The field shall contain one of the various types of shader nodes as specified in 31 Programmable shaders component.

12.4.3 FillProperties

```
FillProperties : X3DAppearanceChildNode {
  SFBool  [in,out] filled     TRUE
  SFColor [in,out] hatchColor 1 1 1 [0,1]
  SFBool  [in,out] hatched    TRUE
  SFInt32 [in,out] hatchStyle 1     [0,∞)
  SFNode  [in,out] metadata   NULL  [X3DMetadataObject]
}
```

The FillProperties node specifies additional properties to be applied to all polygonal areas on top of whatever appearance is specified by the other fields of the respective Appearance node. Thus, hatches are applied on top of the already rendered appearance of the node. Thus, if *filled* is TRUE, the polygonal area is filled according to the other fields of the Appearance node. If *hatched* is TRUE, the polygonal area is hatched as specified by the *hatchStyle* field. Hatches shall be applied after fills are applied.

The *hatchStyle* field selects a hatch pattern as defined in the International Register of Graphical Items (see 2. [REG]). The hatches are rendered using the colour specified by the *hatchColor* field. Browsers shall support hatchstyles 1-6 with hatchstyle 1 being the default. X3D browsers may support any other of the registered hatchstyles. If a hatchstyle that is not supported is requested, hatchstyle 1 shall be used. Table 12.2 specifies the first nineteen hatch styles as defined in the Hatchstyle Section of the International Register of Items. Examples of each hatch style are available at the International Register of Items.

Table 12.2 — International register of items hatchstyles

1	Horizontal equally spaced parallel lines
2	Vertical equally spaced parallel lines
3	Positive slope equally spaced parallel lines
4	Negative slope equally spaced parallel lines
5	Horizontal/vertical crosshatch
6	Positive slope/negative slope crosshatch
7	(cast iron or malleable iron and general use for all materials)

8	(steel)
9	(bronze, brass, copper, and compositions)
10	(white metal, zinc, lead, babbit, and alloys)
11	(magnesium, aluminum, and aluminum alloys)
12	(rubber, plastic, and electrical insulation)
13	(cork,felt, fabric, leather, and fibre)
14	(thermal insulation)
15	(titanium and refractory material)
16	(marble, slate, porcelain, glass, etc.)
17	(earth)
18	(sand)
19	(repeating dot)

The associated geometry shall be filled and/or hatched only when the respective values of the *filled* and/or *hatched* fields have value TRUE.

12.4.4 LineProperties

```
LineProperties : X3DAppearanceChildNode {
  SFBool  [in,out] applied              TRUE
  SFInt32 [in,out] linetype             1    [1,∞)
  SFFloat [in,out] linewidthScaleFactor 0    (-∞,∞)
  SFNode  [in,out] metadata             NULL [X3DMetadataObject]
}
```

The LineProperties node specifies additional properties to be applied to all line geometry. The *linetype* and ~~*linewidth*~~ *linewidthScaleFactor* fields shall only be applied when the *applied* field has value TRUE. When the value of the *applied* field is FALSE, a solid line of nominal width shall be produced. The colour of the line is specified by the associated Material node or X3DColorNode color values.

The *linetype* field selects a line pattern as defined in the International Register of Graphical Items (see 2. [REG]). X3D browsers shall support *linetype* values 1 through 5, with 1 being the default value. X3D browsers may support any other of the registered *linetype* values. If a *linetype* that is not supported is requested, value 1 shall be used. Table 12.2 specifies the first sixteen *linetype* values as defined in the Linetype Section of the International Register of Items.

Table 12.3 — International register of items linetypes

1	Solid
2	Dashed
3	Dotted
4	Dashed-dotted
5	Dash-dot-dot
6	(single arrow)
7	(single dot)
8	(double arrow)
10	(chain line)
11	(center line)
12	(hidden line)
13	(phantom line)
14	(break line 1)
15	(break line 2)
16	User-specified dash pattern

The arrowhead is drawn as short lines forming barbs at any convenient angle between 15 and 90 degrees. The arrowhead is closed and filled in. For linetype "single arrow", the arrowhead is rendered so that the arrow tip occurs at the last point of the each individual list of points passed to a polyline and is in the direction of the last vector. For linetype "double arrow", the first arrowhead is rendered so that the arrow tip occurs at the first point of the list of points passed to a polyline and is in the reverse direction of the first vector. The second arrowhead is rendered as for "single arrow" at the opposite end of the polyline.

The *linewidthScaleFactor* field is a multiplicative value that scales a browser-dependent nominal line width by the given value. This resulting value shall then be mapped to the nearest available line width. A value less than or equal to zero refers to the minimum available line width.

12.4.5 Material

```
Material : X3DOneSidedMaterialNode {
  SFFloat  [in,out] ambientIntensity          0.2          [0,1]
  SFNode   [in,out] ambientTexture            NULL         [X3DSingleTextureNode]
  SFString [in,out] ambientTextureMapping     ""

  SFColor  [in,out] diffuseColor              0.8 0.8 0.8  [0,1]
  SFNode   [in,out] diffuseTexture            NULL         [X3DSingleTextureNode]
  SFString [in,out] diffuseTextureMapping     ""

  SFColor  [in,out] emissiveColor             0 0 0        [0,1]
  SFNode   [in,out] emissiveTexture           NULL         [X3DSingleTextureNode]
  SFString [in,out] emissiveTextureMapping    ""

  SFNode   [in,out] metadata                  NULL         [X3DMetadataObject]

  SFFloat  [in,out] normalScale               1            [0, ∞]
  SFNode   [in,out] normalTexture             NULL         [X3DSingleTextureNode]
  SFString [in,out] normalTextureMapping      ""

  SFFloat  [in,out] occlusionStrength         1            [0,1]
  SFNode   [in,out] occlusionTexture          NULL         [X3DSingleTextureNode]
  SFString [in,out] occlusionTextureMapping   ""

  SFFloat  [in,out] shininess                 0.2          [0,1]
  SFNode   [in,out] shininessTexture          NULL         [X3DSingleTextureNode]
  SFString [in,out] shininessTextureMapping   ""

  SFColor  [in,out] specularColor             0 0 0        [0,1]
  SFNode   [in,out] specularTexture           NULL         [X3DSingleTextureNode]
  SFString [in,out] specularTextureMapping    ""

  SFFloat  [in,out] transparency              0            [0,1]
}
```

~~The Material node specifies surface material properties for associated geometry nodes and is used by the X3D lighting equations during rendering. 17 Lighting component contains a detailed description of the X3D lighting~~

~~model equations.~~

~~All of the fields in the Material node range from 0.0 to 1.0.~~

~~The fields in the Material node determine how light reflects off an object to create colour:~~

a. ~~The *ambientIntensity* field specifies how much ambient light from light sources this surface shall reflect. Ambient light is omnidirectional and depends only on the number of light sources, not their positions with respect to the surface. Ambient colour is calculated as *ambientIntensity* × *diffuseColor*.~~
b. ~~The *diffuseColor* field reflects all X3D light sources depending on the angle of the surface with respect to the light source. The more directly the surface faces the light, the more diffuse light reflects.~~
c. ~~The *emissiveColor* field models "glowing" objects. This can be useful for displaying pre-lit models (where the light energy of the room is computed explicitly), or for displaying scientific data.~~
d. ~~The *specularColor* and *shininess* fields determine the specular highlights (*e.g.*, the shiny spots on an apple). When the angle from the light to the surface is close to the angle from the surface to the viewer, the *specularColor* is added to the diffuse and ambient colour calculations. Lower shininess values produce soft glows, while higher values result in sharper, smaller highlights.~~
e. ~~The *transparency* field specifies how "clear" an object is, with 1.0 being completely transparent, and 0.0 completely opaque.~~

The Material node specifies surface material properties for associated geometry nodes. It indicates that a surface is using *Phong lighting model*. 17 Lighting component contains a detailed description of the X3D lighting model equations.

The material parameters are specified as scalars or RGB colors in the X3D file. All of the `SFFloat` and `SFColor` fields in the Material node range from 0.0 to 1.0.

Moreover every material parameter can be adjusted using a texture. This allows to vary this parameter across the surface. The information sampled from the texture is always multiplied by the simple scalar/color fields.

Examples of texture usage:

- Texture assigned to the *diffuseTexture* controls the most intuitive "visible color of the object". This is the most often used texture.
- Texture assigned to the *specularTexture* allows the surface to be partially shiny (white values in the texture) and partially matte (black values in the texture).

The fields in the Material node determine how light reflects off an object to create color:

a. The *ambientIntensity* and *ambientTexture* fields specify how much ambient light from light sources this surface shall reflect. Ambient light is omnidirectional and depends only on the number of light sources, not their positions with respect to the surface.

   Ambient parameter is calculated as *ambientIntensity* × *diffuseColor* × *textureSample(ambientTexture).rgb*.

b. The *diffuseColor* and *diffuseTexture* fields reflect all X3D light sources depending on the angle of the surface with respect to the light source. The more directly the surface faces the light, the more diffuse light reflects.
c. The *emissiveColor* and *emissiveTexture* fields model "glowing" or "unlit" objects. See X3DOneSidedMaterialNode for the description of these fields.
d. The *specularColor*, *specularTexture*, *shininess* and *shininessTexture* fields determine the specular highlights (*e.g.*, the shiny spots on an apple).

   When the angle from the light to the surface is close to the angle from the surface to the viewer, the *specularColor* × *textureSample(specularTexture).rgb* is added to the diffuse and ambient color calculations.

   Lower shininess values produce soft glows, while higher values result in sharper, smaller highlights. Shininess is calculated as *shininess* × *textureSample(shininessTexture).a*.

e. The *transparency* field (together with alpha channel of the *diffuseTexture*) specifies how "clear" an object is, with 1.0 being completely transparent, and 0.0 completely opaque.

   The transparency determines the *opacity* as `opacity = 1.0 - transparency`. This is then multiplied by the alpha channel of *diffuseTexture* to determine the final alpha of the rendered pixel.

The RGB channels of *diffuseTexture*, *specularTexture* and *emissiveTexture* are multiplied by the corresponding *diffuseColor*, *specularColor*, *emissiveColor* before being used in the current lighting calculation. The alpha channel of a *diffuseTexture* is multiplied by the material *opacity* (which equals just `1.0 - transparency`). The alpha channels contents of *specularTexture* and *emissiveTexture* are ignored.

The *shininessTexture* alpha channel contains values multiplied with the *shininess* factor of the Material node. The RGB channels contents of the *shininessTexture* are ignored.

It is expected, and advised, that authors reuse the same texture node for *specularTexture* and *shininessTexture*. The specular data is deliberately contained in different channels (RGB) than the shininess data (Alpha).

The optional *occlusionTexture* can be used to indicate areas of indirect lighting, typically called *ambient occlusion*. Only the *Red* channel of the texture is used for the computation, the other channels are ignored. Higher values indicate areas that should receive full indirect lighting and lower values indicate no indirect lighting. The *occlusionStrength* determines how much does the occlusion texture affect the final result.

See the section 12.2.4 Texture mapping specified in material nodes for a description how the texture coordinates and texture coordinate transformations are determined based on the *xxxTextureMapping* fields of this node.

12.4.6 PhysicalMaterial

```
PhysicalMaterial : X3DOneSidedMaterialNode {
  SFColor  [in,out] baseColor                     1 1 1  [0,1]
  SFNode   [in,out] baseTexture                   NULL   [X3DSingleTextureNode]
  SFString [in,out] baseTextureMapping            ""

  SFColor  [in,out] emissiveColor                 0 0 0  [0,1]
  SFNode   [in,out] emissiveTexture               NULL   [X3DSingleTextureNode]
  SFString [in,out] emissiveTextureMapping        ""

  SFNode   [in,out] metadata                      NULL   [X3DMetadataObject]

  SFFloat  [in,out] metallic                      1      [0,1]
  SFNode   [in,out] metallicRoughnessTexture      NULL   [X3DSingleTextureNode]
  SFString [in,out] metallicRoughnessTextureMapping ""

  SFFloat  [in,out] normalScale                   1      [0, ∞]
  SFNode   [in,out] normalTexture                 NULL   [X3DSingleTextureNode]
  SFString [in,out] normalTextureMapping          ""

  SFFloat  [in,out] occlusionStrength             1      [0,1]
  SFNode   [in,out] occlusionTexture              NULL   [X3DSingleTextureNode]
  SFString [in,out] occlusionTextureMapping       ""

  SFFloat  [in,out] roughness                     1      [0,1]

  SFFloat  [in,out] transparency                  0      [0,1]
}
```

The PhysicalMaterial node specifies surface material properties for associated geometry nodes. It indicates that a physical lighting model should be used for the computation. 17 Lighting component contains a detailed description of the X3D lighting model equations.

The physical lighting equation, as an input, relies on the following parameters:

- *baseParameter* (RGB color) is, in simple cases, a multiplication of *baseTexture* RGB channel (if such texture was specified) with the *baseColor*.

In other words, it is calculated at every pixel as *baseColor* × *textureSample(baseTexture).rgb*.

Note: This interpretation is true in most cases, but in general it is a simplification of what actually happens. The texture may also come from *Appearance.texture*, and it can even be *MultiTexture* in which case it is not necessarily multiplied. See the 17.2.2.6 Physical lighting model for the exact specification how the *baseParameter* is calculated in every possible case.

- *metallicParameter* is a multiplication of *metallic* with the *Blue* texture channel of *metallicRoughnessTexture* (if such texture was specified).

  In other words, it is calculated at every pixel as *metallic* × *textureSample(metallicRoughnessTexture).b*.

- *roughnessParameter* is a multiplication of *roughness* with the *Green* texture channel of *metallicRoughnessTexture* (if such texture was specified).

  In other words, it is calculated at every pixel as *roughness* × *textureSample(metallicRoughnessTexture).g*.

When calculating *metallicParameter* and *roughnessParameter* terms, the *Red* and *Alpha* channels of the *metallicRoughnessTexture* are ignored. It is possible to use the same texture for *metallicRoughnessTexture* and *occlusionTexture*, as they deliberately look at different channels, so all the information can be contained in one RGB texture.

The final alpha, used for blending or alpha-testing, is calculated as *baseTexture* alpha channel multiplied with the opacity (`1.0 - transparency`). This is consistent with the behavior of *diffuseColor*, *diffuseTexture* and *transparency* on the Phong Material. If the *baseTexture* was not specified, it is also possible to use the *Application.texture*. See the 17.2.2.6 Physical lighting model for the exact specification, and the 12.2.5 Coexistence of textures specified in material nodes with the "Appearance.texture" field for a description how *Appearance.texture* is used.

Moreover the PhysicalMaterial defines the *emissiveColor* and optional *emissiveTexture*. The resulting *emissiveParameter* term is simply added to the pixel color, this behavior is consistent for all X3D materials.

The optional *occlusionTexture* can be used to indicate areas of indirect lighting, typically called *ambient occlusion*. Only the *Red* channel of the texture is used for the computation, the other channels are ignored. Higher values indicate areas that should receive full indirect lighting and lower values indicate no indirect lighting. The *occlusionStrength* determines how much does the occlusion texture affect the final result.

Physical interpretation of the material parameters:

Note: The physical material properties of X3D are deliberately consistent with the glTF 2.0 material definition. Effectively, converting between (in both directions) between X3D PhysicalMaterial and glTF 2.0 material definitions is trivial.

The description of the parameter meaning below follows very closely the glTF specification.

The *baseParameter* color has two different interpretations depending on the value of *metallicParameter*. When the material is a metal, the *baseParameter* color is the specific measured reflectance value at normal incidence (F_0). For a non-metal the *baseParameter* color represents the reflected diffuse color of the material. In this model it is not possible to specify a F_0 value for non-metals, and a linear value of 4% (0.04) is used.

The following equations show how to calculate bidirectional reflectance distribution function (BRDF) inputs (c_{diff}, F_0, α) from the metallic-roughness material properties.

```
const dielectricSpecular = rgb(0.04, 0.04, 0.04)
const black = rgb(0, 0, 0)
c_diff = lerp(baseParameter * (1 - dielectricSpecular.r), black, metallicParameter)
F_0 = lerp(dieletricSpecular, baseParameter, metallicParameter)
α = roughnessParameter ^ 2
```

See the section 12.2.4 Texture mapping specified in material nodes for a description how the texture coordinates and texture coordinate transformations are determined based on the *xxxTextureMapping* fields of this node.

12.4.7 PointProperties

```
PointProperties : X3DAppearanceChildNode {
  SFFloat  [in,out] pointSizeScaleFactor  1 [1,∞)
  SFFloat  [in,out] pointSizeMinValue     1 [0,∞)
  SFFloat  [in,out] pointSizeMaxValue     1 [0,∞)
  SFVec3f  [in,out] attenuation           1 0 0 [0,∞)
  SFString [in,out] colorMode             "TEXTURE_AND_POINT_COLOR" ["POINT_COLOR" | "TEXTURE_COLOR" | "TEXTURE_AND_POINT_COLOR"]
  SFNode   [in,out] metadata              NULL [X3DMetadataObject]
}
```

The PointProperties node specifies additional properties to be applied to all point geometry. The colour of the line is specified by the associated Material node or X3DColorNode color values.

pointSizeScaleFactor is a value determining the nominal point size before modification by the sizing modifications, as determined by the *pointSizeMinValue*, *pointSizeMaxValue*, and *attenuation* values discussed below. The nominal rendered point size is a browser-dependent minimum renderable point size.

pointSizeMinValue is minimum allowed scaling factor on nominal browser point scaling. *pointSizeMaxValue* is maximum allowed scaling factor on nominal browser point scaling. The provided value for *pointSizeMinValue* must be less than or equal to value for *pointSizeMaxValue*.

The *attenuation* field defines a depth perception effect in a point cloud rendering by making points close to the viewer appear larger. The modification of point size depending on distance from the view occurs in two steps, starting with the nominal point size as determined by the *pointSizeScaleFactor* field. The *attenuation* field defines three parameters `a`, `b`, and `c` from the components of a single SFVec3f value:

```
a = attenuation[0]
b = attenuation[1]
c = attenuation[2]
```

Together these parameters define an attenuation factor $1/(a + b \times r + c \times r^2)$ where `r` is the distance from the observer position (current viewpoint) to each point. The nominal point size is multiplied by the attenuation factor and then clipped to a minimum value of *pointSizeMinValue* × the minimum renderable point size, then clipped to a maximum size of *pointSizeMaxValue* × minimum renderable point size.

When a X3DTextureNode is defined in the same Appearance instance as PointProperties node, the points of a PointSet shall be displayed as point sprites using the given texture(s). ~~The *colorMode* field has a blending effect on the rendering of point sprites. A value of:~~

- ~~POINT_COLOR shall display the RGB channels of the color instance defined in X3DMaterialNode or X3DColorNode, and the A channel of the texture if any. If no color is associated to the point, the default RGB color (0, 0, 0) shall be used.~~
- ~~TEXTURE_COLOR shall display the original texture with its RGBA channels and regardless to the X3DMaterialNode or X3DColorNode which might be associated to the point set.~~
- ~~TEXTURE_AND_POINT_COLOR shall display the RGBA channels of a texture added to the RGB channels of the color defined in X3DMaterialNode or X3DColorNode node, and the A channel of the texture if any. If no color is associated to the point, the result shall be exactly the same as TEXTURE_COLOR.~~

~~If no X3DTextureNode is defined in the same Appearance instance as PointProperties, points of the PointSet shall be displayed anti-aliased and using their associated colors.~~

TODO reference/bibliography International register of items for markertype.

12.4.8 Shape

```
Shape : X3DShapeNode {
  SFNode  [in,out] appearance  NULL     [X3DAppearanceNode]
```

```
  SFBool  [in out] bboxDisplay FALSE
  SFNode  [in,out] geometry    NULL     [X3DGeometryNode]
  SFNode  [in,out] metadata    NULL     [X3DMetadataObject]
  SFBool  [in out] visible     TRUE
  SFVec3f []       bboxCenter  0 0 0    (-∞,∞)
  SFVec3f []       bboxSize    -1 -1 -1 [0,∞) or -1 -1 -1
}
```

The Shape node has two fields, *appearance* and *geometry,* that are used to create rendered objects in the world. The *appearance* field contains an Appearance node that specifies the visual attributes (*e.g.*, material and texture) to be applied to the geometry. The *geometry* field contains a geometry node. The specified geometry node is rendered with the specified appearance nodes applied. See 12.2 Concepts for more information.

17 Lighting component contains details of the X3D lighting model and the interaction between Appearance nodes and geometry nodes.

If the *geometry* field is NULL, the object is not drawn.

The *bboxCenter* and *bboxSize* fields specify a bounding box that encloses the Shape node's geometry. This is a hint that may be used for optimization purposes. The results are undefined if the specified bounding box is smaller than the actual bounding box of the geometry at any time. A default *bboxSize* value, (-1 -1 -1), implies that the bounding box is not specified and, if needed, is calculated by the browser. A description of the *bboxCenter* and *bboxSize* fields is contained in 10.2.2 Bounding boxes.

12.4.9 TwoSidedMaterial (deprecated)

```
TwoSidedMaterial : X3DMaterialNode {
  SFFloat [in,out] ambientIntensity     0.2         [0,1]
  SFFloat [in,out] backAmbientIntensity 0.2         [0,1]
  SFColor [in,out] backDiffuseColor     0.8 0.8 0.8 [0,1]
  SFColor [in,out] backEmissiveColor    0 0 0       [0,1]
  SFFloat [in,out] backShininess        0.2         [0,1]
  SFColor [in,out] backSpecularColor    0 0 0       [0,1]
  SFFloat [in,out] backTransparency     0           [0,1]
  SFColor [in,out] diffuseColor         0.8 0.8 0.8 [0,1]
  SFColor [in,out] emissiveColor        0 0 0       [0,1]
  SFNode  [in,out] metadata             NULL        [X3DMetadataObject]
  SFFloat [in,out] shininess            0.2         [0,1]
  SFBool  [in,out] separateBackColor    FALSE
  SFColor [in,out] specularColor        0 0 0       [0,1]
  SFFloat [in,out] transparency         0           [0,1]
}
```

This node is deprecated since X3D version 4.0. Future versions of the standard may remove this node. The upgrade path is as follows:

- If you used *TwoSidedMaterial* with *separateBackColor* equal FALSE (default), then simply use the simpler Material node instead. In case of *separateBackColor* equal FALSE, the *TwoSidedMaterial* was actually useless. Only the *solid* field, described in the 11.2.3 Common geometry fields, controls whether the geometry is visible from the back side (this was true in X3D 3.x and remains true in X3D 4.x).
- If you used *TwoSidedMaterial* with *separateBackColor* equal TRUE, then instead use *Appearance.backMaterial* field to specify different rendering parameters for the back faces. See Two-sided materials.

This node defines material properties that can effect both the front and back side of a polygon individually. These materials are used for both the front and back side of the geometry whenever the X3D lighting model is active.

If the *separateBackColor* field is set to TRUE, the rendering shall render the front and back faces of the geometry with different values. If the value is FALSE, the front colours are used for both the front and back side of the polygon, as per the existing X3D lighting rules.

When calculating the terms in the lighting equations, the front geometry shall use the fields *ambientIntensity*, *diffuseColor*,*emissiveColor*, *shininess*, *specularColor*, and *transparency*. The faces that are determined to be the back side are rendered using *backAmbientIntensity*, *backDiffuseColor*, *backEmissiveColor*, *backShininess*, and *backTransparency* as the appropriate components in the lighting equations.

12.4.10 UnlitMaterial

```
UnlitMaterial : X3DOneSidedMaterialNode {
  SFColor  [in,out] emissiveColor                1 1 1  [0,1]
  SFNode   [in,out] emissiveTexture              NULL   [X3DSingleTextureNode]
  SFString [in,out] emissiveTextureMapping       ""
  SFNode   [in,out] metadata                     NULL   [X3DMetadataObject]
  SFFloat  [in,out] normalScale                  1      [0, ∞]
  SFNode   [in,out] normalTexture                NULL   [X3DSingleTextureNode]
  SFString [in,out] normalTextureMapping         ""
  SFFloat  [in,out] transparency                 0      [0,1]
}
```

Material that is unaffected by light sources. Suitable to create various non-realistic effects, when the colors are defined explicitly and are not affected by the placement of the shape relative to the lights or camera.

The output color and opacity, called *emissiveParameter* by the lighting equations, are determined like this:

1. Use the *emissiveColor* field value as the *emissiveParameter.rgb*. Use the *1.0 - transparency* as the *emissiveParameter.a*.

2. If shape is using Color node then the information from Color node overrides the *emissiveParameter.rgb*. If shape is using ColorRGBA node then the information from ColorRGBA overrides both the *emissiveParameter.rgb* and the *emissiveParameter.a*.

   Note: This is consistent with how Color or ColorRGBA override diffuseColor and transparency in case of Material.

3. If the *emissiveTexture* is not NULL, then it multiplies (component-wise) the *emissiveParameter.rgb* (multiplied by the texture RGB channels) and *emissiveParameter.a* (multiplied by the texture alpha channel).

   If the *emissiveTexture* is NULL, but *Appearance.texture* field is not NULL, then the same logic is applied to the *Appearance.texture* texture: it multiplies *emissiveParameter.rgb* and *emissiveParameter.a*. See 12.2.5 Coexistence of textures specified in material nodes with the "Appearance.texture" field for a description how is the *Appearance.texture* field used.

Note about default values: This node inherits the *emissiveColor* field from the X3DOneSidedMaterialNode ancestor, but the default value of this field changes: only for UnlitMaterial, the default *emissiveColor* is `1 1 1` (white), instead of `0 0 0` (black, default of *X3DOneSidedMaterialNode.emissiveColor*).

Implementation hint: Normal vectors information is not useful for the calculation of unlit material. Implementations can ignore the normal vectors provided in the geometry node (per-face or per-vertex) and in the *normalTexture* field. Implementations are encouraged to optimize this case, and not send unneeded normals data to GPU, and not calculate implicit normal vectors (normally derived from creaseAngle and ccw fields). *However, there is an exception to this optimization*: if the shape is using TextureCoordinateGenerator with some modes (`CAMERASPACENORMAL`, `CAMERASPACEREFLECTIONVECTOR`) then the shader code may need access to normals anyway.

See the section 12.2.4 Texture mapping specified in material nodes for a description how the texture coordinates and texture coordinate transformations are determined based on the *xxxTextureMapping* fields of this node.

12.5 Support levels

The Shape component provides three levels of support as specified in Table 12.4.

Table 12.4 — Shape component support levels

Level	Prerequisites	Nodes/Features	Support
1	Core 1 Rendering 1 Texturing 1		
		X3DAppearanceChildNode(abstract)	n/a
		X3DAppearanceNode (abstract)	n/a
		X3DMaterialNode (abstract)	n/a
		X3DOneSidedMaterialNode (abstract)	n/a
		X3DShapeNode (abstract)	n/a
		Appearance	Optional support for *textureTransform*, *lineProperties*, *fillProperties*, *shaders*, *backMaterial*.
		Material	Optional support for *ambientIntensity*, *shininess*, *specularColor* and all *xxxTexture...* fields except *diffuseTexture* (that is, optional support for: *ambientTexture*, *emissiveTexture*, *normalTexture*, *occlusionTexture*, *shininessTexture*, *specularTexture*).
		UnlitMaterial	All fields fully supported.
		Shape	All fields fully supported.
2	Core 1 Lighting 4 Rendering 1 Texturing 1		
		All Level 1 nodes except Appearance	All fields fully supported.

		Appearance	Optional support for the same properties as on level 1.
		LineProperties	All fields fully supported.
		PhysicalMaterial	All fields fully supported.
3	Core 1 Lighting 4 Rendering 1 Texturing 1		
		All Level 2 nodes except Appearance	All fields fully supported.
		Appearance	Optional support for *backMaterial*.
		FillProperties	All fields fully supported.
4	Core 1 Lighting 4 Grouping 1 Rendering 1		
		All Level 3 nodes	All fields fully supported.
		PointProperties	All fields fully supported.
		TwoSidedMaterial (deprecated)	Support optional.
5	Core 1 Lighting 4 Grouping 1 Rendering 1		
		All Level 4 nodes	All fields fully supported.
		AcousticProperties	All fields fully supported.