

Australian Standard™

**Information technology—Coding of
audio-visual objects**

Part 19: Synthesized texture stream



This Australian Standard was prepared by Committee IT-029, Coded Representation of Picture, Audio and Multimedia/Hypermedia Information. It was approved on behalf of the Council of Standards Australia on 27 October 2004. This Standard was published on 30 November 2004.

The following are represented on Committee IT-029:

Australian Broadcasting Authority
Australian Broadcasting Corporation
Australian Consumer Association
Australian Subscription Television
CSIRO Information and Communication Technology Centre
DSTC
Department of Defence (Australia)
Free TV Australia
Special Broadcasting Service
The University of New South Wales
University of Sydney
University of Wollongong
Victoria University of Technology

Keeping Standards up-to-date

Standards are living documents which reflect progress in science, technology and systems. To maintain their currency, all Standards are periodically reviewed, and new editions are published. Between editions, amendments may be issued. Standards may also be withdrawn. It is important that readers assure themselves they are using a current Standard, which should include any amendments which may have been published since the Standard was purchased.

Detailed information about Standards can be found by visiting the Standards Web Shop at www.standards.com.au and looking up the relevant Standard in the on-line catalogue.

Alternatively, the printed Catalogue provides information current at 1 January each year, and the monthly magazine, *The Global Standard*, has a full listing of revisions and amendments published each month.

Australian Standards™ and other products and services developed by Standards Australia are published and distributed under contract by SAI Global, which operates the Standards Web Shop.

We also welcome suggestions for improvement in our Standards, and especially encourage readers to notify us immediately of any apparent inaccuracies or ambiguities. Contact us via email at mail@standards.org.au, or write to the Chief Executive, Standards Australia International Ltd, GPO Box 5420, Sydney, NSW 2001.

This Standard was issued in draft form for comment as DR 04419.

Australian Standard™

Information technology—Coding of audio-visual objects

Part 19: Synthesized texture stream

First published as AS ISO/IEC 14496.19—2004.

COPYRIGHT

© Standards Australia International

All rights are reserved. No part of this work may be reproduced or copied in any form or by any means, electronic or mechanical, including photocopying, without the written permission of the publisher.

Published by Standards Australia International Ltd GPO Box 5420, Sydney, NSW 2001, Australia

ISBN 0 7337 6372 3

PREFACE

This Standard was prepared by the Standards Australia Committee IT-029, Coded Representation of Picture, Audio and Multimedia/Hypermedia Information.

This Standard is identical with, and has been reproduced from, ISO/IEC 14496-19:2004, *Information technology—Coding of audio-visual objects—Part 19: Synthesized texture stream*.

The objective of this Standard is to provide designers of multimedia content with guidance for creating synthesized texture streams for very low bit rate synthetic video clips, within the MPEG-4 encoding audio-visual presentation.

The terms ‘normative’ and ‘informative’ are used to define the application of the annex to which they apply. A normative annex is an integral part of a standard, whereas an informative annex is only for information and guidance.

As this Standard is reproduced from an international standard, the following applies:

- (a) Its number appears on the cover and title page while the international standard number appears only on the cover.
- (b) In the source text ‘this International Standard’ should read ‘this Australian Standard’.
- (c) A full point substitutes for a comma when referring to a decimal marker.

References to International Standards should be replaced by references to Australian or Australian/New Zealand Standards, as follows:

<i>Reference to International Standard</i>		<i>Australian Standard</i>	
ISO/IEC		AS/NZS	
14496	Information technology—Coding of audio-visual objects	14496	Information technology—Coding of audio-visual objects
14496-1	Part 1: Systems	14496.1	Part 1: Systems

Only referenced documents that have been adopted as Australian or Australian/New Zealand Standards have been listed.

CONTENTS

Page

Introduction	iv
1 Scope.....	1
2 Normative References	1
3 Synthesized Texture Compression Technology.....	1
3.1 Functionality and Semantics	1
4 Coding and Bitstream.....	46
4.1 Overview	46
4.2 Global Input Bitstream and Decoding Context	46
4.3 Header Block ('H') Decoding	48
4.4 Scene Block ('S') Decoding.....	49
4.5 Object Block ('C') Decoding	50
4.6 Texture Block ('A') Decoding	51
4.7 Skeleton Decoding	73
4.8 Animation Decoding	76
4.9 Camera Decoding.....	80
4.10 Quantization	81
4.11 Sub-Streams	82
5 SynthesizedTexture Data Stream	86
5.1 Structure of the SynthesizedTexture Data Stream	86
5.2 Access Unit Definition	86

INTRODUCTION

ISO/IEC 14496 specifies a system for the communication of interactive audio-visual scenes. The specification includes the following elements:

1. the coded representation of natural or synthetic, two-dimensional (2D) or three-dimensional (3D) objects that can be manifested audibly and/or visually (audio-visual objects) (specified in part 1,2 and 3 of ISO/IEC 14496);
2. the coded representation of the spatio-temporal positioning of audio-visual objects as well as their behavior in response to interaction (scene description, specified in part 11 of ISO/IEC 14496);
3. the coded representation of information related to the management of data streams (synchronization, identification, description and association of stream content, specified in part 11 of ISO/IEC 14496);
4. a generic interface to the data stream delivery layer functionality (specified in part 6 of ISO/IEC 14496);
5. an application engine for programmatic control of the player: format, delivery of downloadable Java byte code as well as its execution lifecycle and behavior through APIs (specified in part 11 of ISO/IEC 14496); and
6. a file format to contain the media information of an ISO/IEC 14496 presentation in a flexible, extensible format to facilitate interchange, management, editing, and presentation of the media.

The information representation, specified in ISO/IEC 14496-1 and in ISO/IEC 14496-11, describes the means to create an interactive audio-visual scene in terms of coded audio-visual information and associated scene description information. The encoded content is presented to a terminal as the collection of elementary streams. Elementary streams contain the coded representation of either audio or visual data or scene description information or user interaction data. Elementary streams may as well themselves convey information to identify streams, to describe logical dependencies between streams, or to describe information related to the content of the streams. Each elementary stream contains only one type of data.

Elementary streams are decoded using their respective stream-specific decoders. The audio-visual objects are composed according to the scene description information and presented by the terminal's presentation device(s). All these processes are synchronized according to the systems decoder model (SDM) using the synchronization information provided at the synchronization layer.

The scene description stream identifies different types of objects, such as audio, visual, 2D and 3D graphics, etc. that define a scene composition of the content. Synthesized Textures streams provide for photo-realistic animations that can be transmitted using very low bitrates. These type of animations can be used in combination with other streams to enhance any scene.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of patents.

The ISO and IEC take no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured the ISO and IEC that he is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with the ISO and IEC. Information may be obtained from:

Vimatix Inc.
5 Oppenheimer St.
Rehovot 76701
Israel

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

AUSTRALIAN STANDARD

Information technology — Coding of audio-visual objects —

Part 19:

Synthesized texture stream

1 Scope

This part of ISO/IEC 14496 specifies functionalities for the transmission of Synthesized Texture data as part of the MPEG-4 encoded audio-visual presentation. More specifically, it defines:

1. The synthesized texture format representation that is utilized for Synthesized Texture data encoding
2. The coded representation of Synthesized Texture data streams.

2 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 14496-1, *Information technology — Coding of audio-visual objects — Part 1: Systems*

ISO/IEC 14496-11, *Information technology — Coding of audio-visual objects — Part 11: Scene description and application engine*

3 Synthesized Texture Compression Technology

3.1 Functionality and Semantics

3.1.1 Overview

Synthesized Textures represent photo-realistic textures by describing color information of vectors. Synthesized Texture streams are used for creation of very low bit rate synthetic video clips. Synthesized Texture clips are built using key frame based animations of skeletons that affect photorealistic textures whose color information is modeled by equations.

A texture top-level **Synthesized Texture Node (STNode)** can be defined for playing **SynthesizedTextures**, see ISO/IEC 14496-11 for additional details . The **STNode** itself is similar to the **MovieTexture**, and uses url field to reference an Object Descriptor describing the associated stream(s). The stream contains both the object textures and their animation descriptions . The **STNode** also exposes control points that can be used to manipulate via affine transforms the objects carried in its associated stream. By this way **STNode** can implement synthesized interactive SynthesizedTextures. As any texture, the resulting texture can be mapped onto any 2D or 3D surface.

3.1.1.1 SynthesizedTexture Elements

The **SynthesizedTexture** is a collection of animated **Objects** (also called **Actors**) sharing a common Stage, Camera and Timeline.

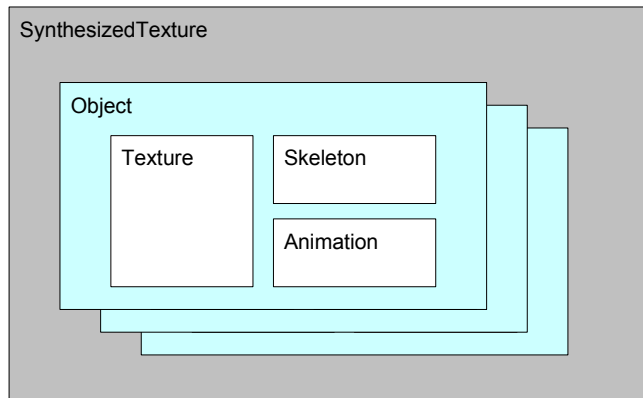


Figure 1 — Synthesized Texture structure

The **Object** is comprised of a Texture, A Skeleton and an Animation.

- The object's **Texture** represents the objects skin.

The texture is comprised of primitive vector-style entities, belonging to a small number of primitive types such as Lines and Area Color Points.

The pixel representation of the texture is reconstructed through the process of **Texture Rendering**.

The texture is divided into mutually exclusive sub-textures called **Layers**.

- The **Skeleton** represents the kinematic capabilities of the object relative to itself and controls the shape and appearance of the skin.

The skeleton is comprised of a topology of **Bones** whose geometric configuration is controlled by the object's animation.

The skeleton is attached to the texture's layers, and controls their position and shape within the object's plane. This ultimately affects the layout of the texture primitives on the plain, as the skeleton geometry changes.

Re-rendering the texture based on a new layout of the texture primitives eventually results in a realistic warping effect called **Texture Warping**.

- The **Animation** represents the spatial behavior of a single object along time.

The animation of Objects is formed by an extrinsic motion of the entire object relative to the world, and an intrinsic motion of Layers relative to the object they are part of.

The intrinsic motion is controlled by the Skeleton geometry, as described above. Extrinsic motion of each object is controlled by its 3D displacement and rotation within the SynthesizedTexture's world.

The Animation is a sequence of **KeyFrames** describing the state of the object in both intrinsic and extrinsic aspects, at specific frames on a timeline. Frames that are not explicitly described by a key-frame are derived by interpolation between neighboring key-frames, in a process called "tweening".

3.1.1.2 SynthesizedTexture Playback

SynthesizedTexture playback is based on animating all the objects in the SynthesizedTexture as described above.

All objects are animated:

- within the SynthesizedTexture's shared 3D world ("**Stage**"),
- across the SynthesizedTexture's shared **Timeline**,
- and optionally also relative to a camera.

The resulting bitmaps rendered from the objects' Textures are projected onto a shared 2D frame buffer, which holds the current raster frame that is ready for display.

3.1.1.3 SynthesizedTexture Coding

The Synthesized Texture **Bitstream** contains the SynthesizedTexture data in a coded form called **ST Coding**.

ST Coding encodes the SynthesizedTexture's primitive entities in a hierarchical compact arrangement resulting in very high compression rates. This is done by employing several techniques:

- a) **Aggregation** – this technique groups and orders specific attributes of primitives into sub-streams, in a manner that is 'friendly' for quantization and packing techniques. For example, location data of all Texture Terminal Points (TPs) and Patches (PAs), are aggregated, ordered according to their "geographic" location, quantized and packed - together. During decoding, sub-stream data are de-multiplexed to their respective primitive types and properties, according to hard-coded rules and soft-coded indicators.
- b) **Quantization** – the numeric values in a sub-stream are rounded, factored, offset or otherwise transformed, focused on reducing stream size, and generating data that is 'friendly' to additional packing techniques. Quantization parameters are typically stored in standard hard-coded quantization tables, so that they do not need to be carried in the bitstream. During decoding, quantized sub-streams are de-quantized using the proper quantization tables, through dequant() methods.
- c) **Packing** – various loss-less compression¹⁾ techniques are applied to further reduce data size. These techniques include variants of Huffman, Run-Length, and other encoding methods. During decoding, sub-streams that were packed are un-packed and decompressed, using the appropriate method.

The ST Bitstream includes four types of top-level blocks: Header, Scene, Objects and Textures. The Object block also contains the Skeleton and/or Animation of the Texture.

ST Coding, including its division to sub-streams, their order, and the manner in which they are encoded, may come in one of several **coding syntaxes**. This specification describes the current default syntax called **coding syntax 0**.

1) The terms **compression** and **decompression** in this document refer to the lossless statistical compression techniques, such as Huffman and Run-Length encoding, used as part of the Synthesized Texture encoding.

Chapter 4 specifies the structure of the ST Bitstream and describes how it is decoded to SynthesizedTexture primitives.

3.1.2 The Texture

3.1.2.1 General

The Texture is used to represent the skin of an object.

The texture is comprised of small set of primitive vector-style entities.

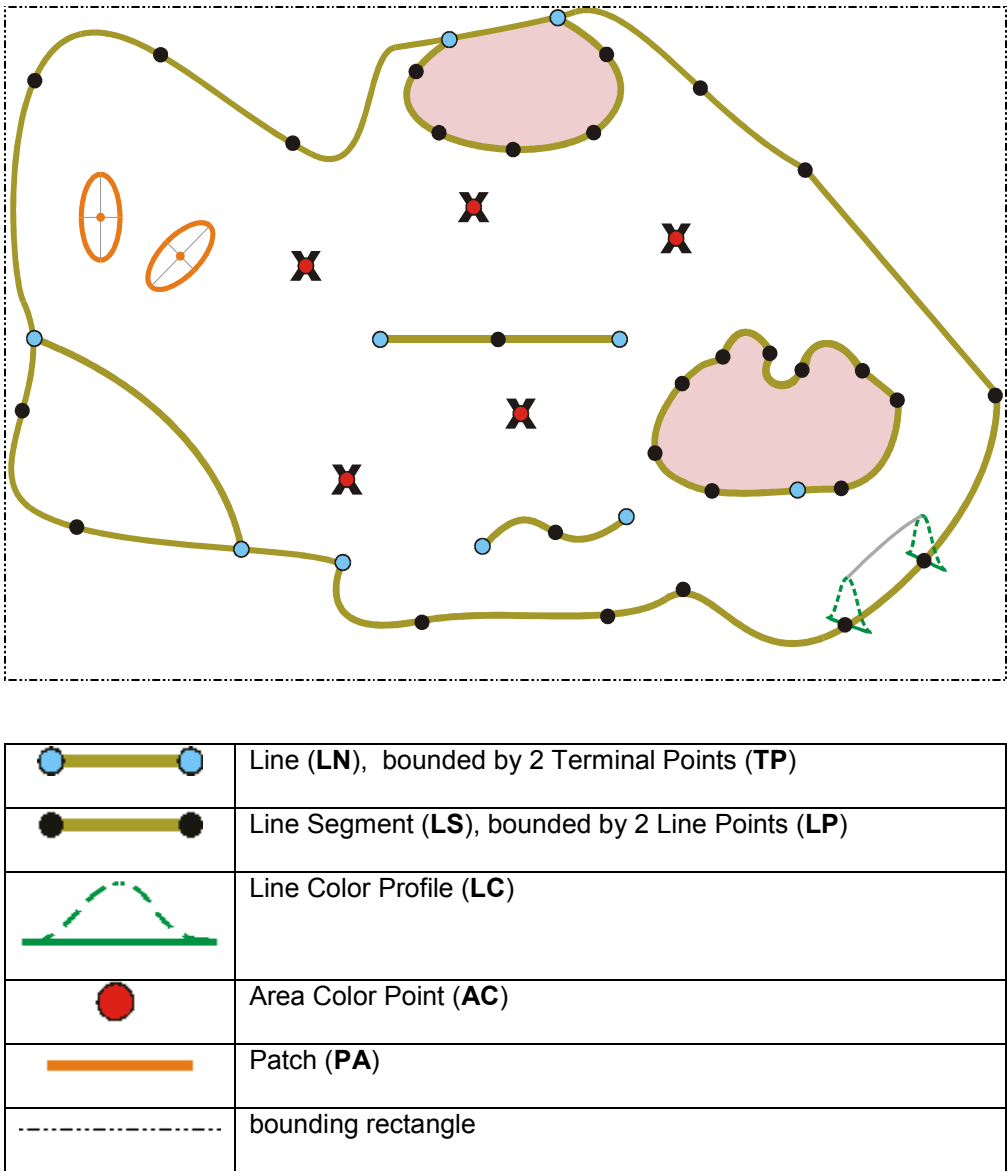


Figure 2 — Synthesized Texture Primitives

3.1.2.2 Primitives and Properties

A Texture is comprised of the following primitive types:

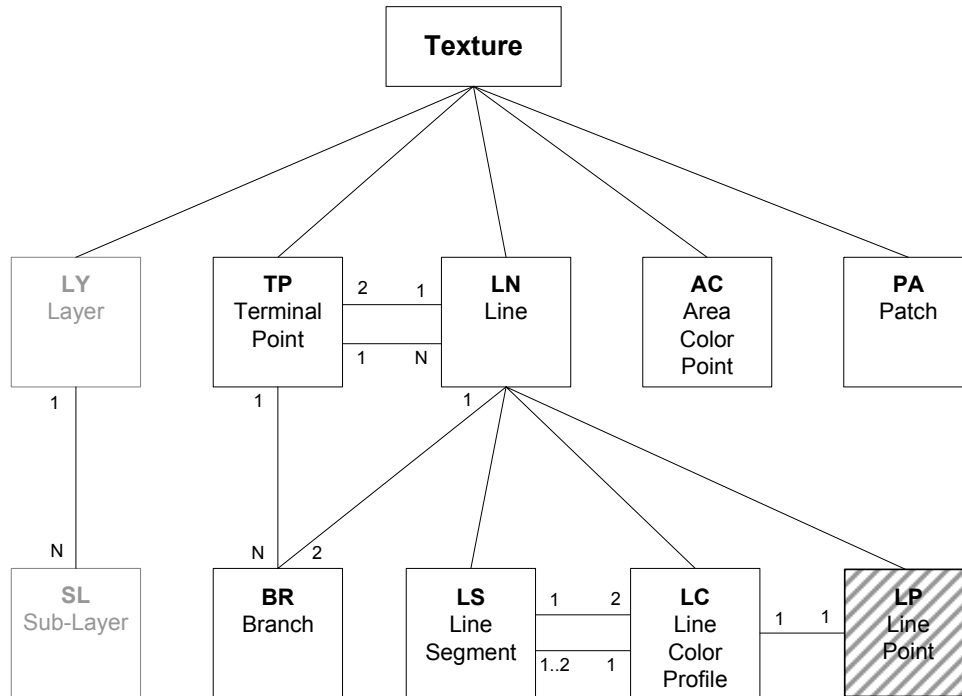


Figure 3 — Synthesized Texture hierarchy of primitives

3.1.2.3 Texture

3.1.2.3.1 Syntax

```

class Texture
{
    int width, height;

    // decoded texture primitives:
    LN aLN[];          // Lines
    TP aTP[];          // Terminal Points
    AC aAC[];          // Area Color Points
    PA aPA[];          // Patches

    int nLN = 0;        // Number of LNs
    int nTP = 0;        // Number of TPs
    int nAC = 0;        // Number of ACs
    int nPA = 0;        // Number of PAs

    int worldUnit = 512;
    float LosOffsetX = 0;
    float LosOffsetY = 0;

    // auxiliary
    LY aLY[];          // Layers
    int nLY = 0;        // Number of LYs
    int nSL = 0;        // total number of sub-layers in texture
}
  
```

3.1.2.3.2 Semantics

Properties:

Name	Description
width, height	The width and height of the texture's bounding rectangle, in pixels. All coordinates of the texture's elements are given relative to the top-left corner of this rectangle.
Ordered arrays of decoded texture primitives:	
aLN	The Lines in the Texture
aTP	The Terminal Points in the Texture
aAC	The Area Color Points in the Texture
aPA	The Patches in the Texture
nLN	The number of elements in aLN
nTP	The number of elements in aTP
nAC	The number of elements in aAC
nPA	The number of elements in aPA
worldUnit	The number of pixels corresponding to 1 "world-unit" in this texture. World units are used to describe distances in the SynthesizedTexture's 3D world. Default: worldUnit = 512 pixels.
LosOffsetX, LosOffsetY	The x,y components of the texture's LOS (Line of Sight) Offset, in pixels. A texture's LOS Offset is the offset of the texture's center from the center of the visual plane of a camera that could have captured the texture. This information allows adapting the appearance of a texture which was photographed from a certain angle (relative to the camera's LOS), to the changing angle in which it is viewed in an animated SynthesizedTexture. For a texture that was photographed from a "straight ahead" angle, the LOS Offset is (0, 0).
Auxiliary:	
aLY	The Layers in the Texture Note layers are an auxiliary structure not used in Texture Rendering; they serve Texture Decoding only for associating Texture primitives to bones in the skeleton.
nLY	The number of elements aLY
nSL	The total number of sub-layers in the Layers in the texture

3.1.2.4 Line (LN)

3.1.2.4.1 Syntax

```
class LN          // Line
{
    int iLnType;
    int iContourType = 0;
    int iSlLft, iSlRgt;

    int iTpBeg, iTpEnd;

    LS  aLS[];
    int nLS;

    LC  aLC[];
    int nLC;
}
```

3.1.2.4.2 Semantics

Lines implement the **characteristic lines** of a Texture. A Line has a geometry described by Line Segments (LSs) and a coloring described by Line Color Profiles (LCs) placed along its course.

A Line primitive is built of a chain of Line Segment (LSs), and its 2D geometry is determined by these segments. Every Line Segment is part of exactly one single Line, and is terminated by exactly 2 Line Points (LPs) – one at each end. In addition to the LPs at the 2 ends of each line, a Line is always terminated by exactly 2 Terminal Points (TP) – one at each end.

Every Line Point hosts exactly 1 Line Color Profile (LC), and every LC is currently hosted by exactly 1 LP. For a closed Line, the number of LSs equals the number of LPs & LCs; For a non-closed Line LSs are 1 less than LPs & LCs.

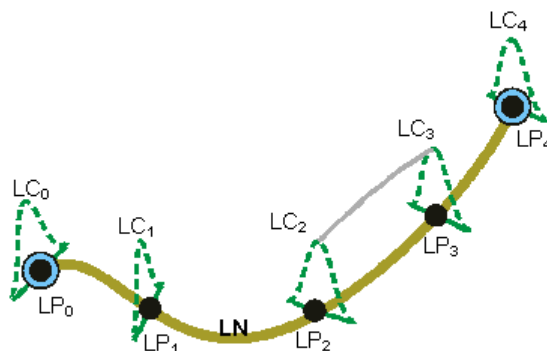


Figure 4 — Synthesized Texture Line (LN)

The collection of LCs along a Line determines the coloring behavior of the line, together with the Line's iLnType.

The external contour of a Texture, and borders between Layers and Sub-Layers are built of Lines. Lines participating in a contour are marked by using the Line's iContourType.

Properties:

Name	Description
iLnType	The type of coloring behavior across the "width" dimension of this Line. This indicates the type of Line Color Profiles (LCs, §3.1.2.8) used along the Line. 00-RIDGE. 01-STRIPE 02-EDGE. 03-ABSENT (Absent out-branch in Splitting). 04-PARALLEL (Parallel Ridge linked to out-branch in Splitting). A distinction is made between separating and non-separating line types. Separating lines block the "diffusion" of color originating at Area Color Points (ACs) near the Line from one side of the Line to the other. RIDGE and EDGE lines are separating, while STRIPE lines are non-separating.
iContourType	Marks whether this Line is part of the contour (i.e. outer border) of the Texture or one of its layers. 00-This Line is not a contour line.
iSILft, iSIRgt	When this Line separates Sub-layers – the indexes of the sub-layers to the left and to the right of the Line.
iTpBeg, iTpEnd	The indexes of the TPs in which this Line begins and ends.
aLS	The array of Line Segments comprising the geometry of this Line.
nLS	The number of elements in aLS. includes last LS ?
aLC	The array of Line Color Profiles describing the coloring of this Line.
nLC	The number of elements in aLC. nLC = nLS+1.

3.1.2.5 Line Segment (LS)

3.1.2.5.1 Syntax

```
class LS                // Line Segment Geometry
{
    float vX, vY;
    float hgt;
}
```

3.1.2.5.2 Semantics

Line Segments are the building blocks of Lines (LNs).

Every Line Segment is part of exactly one Line, and is terminated by exactly 2 Line Points (LPs) – one at each end.

A Line Segment is an arc which follows a globally fixed mathematical model. In the current implementation of the Texture, a Line Segment is unambiguously represented as a parabolic curve. The LS parabola geometry is encoded using its vX, vY and hgt.

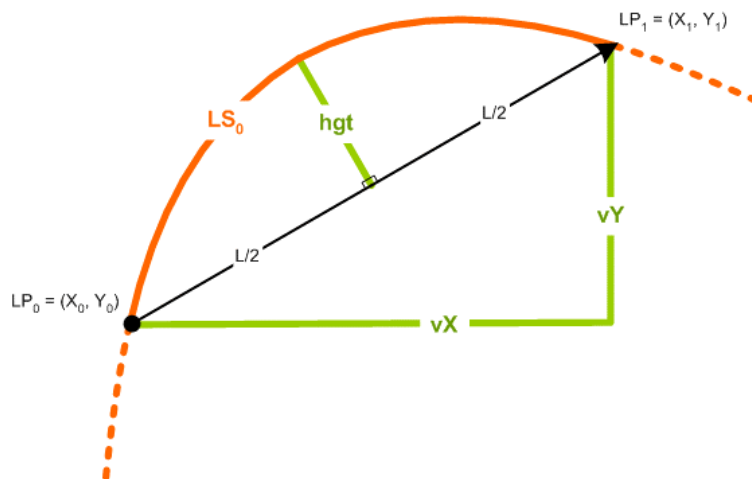


Figure 5 — Synthesized Texture Line Segment (LS)

The base of the LS arc is described by a vector, given by the LS's starting point LP_0 and the (x,y) components of the vector – vX and vY .

The signed height of the LS arc above or below its base is given by $LS.hgt$.

The LS parabola is defined only between the LS's starting point LP_0 and its end point LP_1 .

Properties:

Name	Description
vX, vY	The x,y components of the vector connecting this Line Segment's starting and ending points, in pixel units. The LS curve is defined only between the LS's starting and ending points (LP_0 and LP_1).
hgt	The signed height of the parabola of this LS above or below the LS's vector.

3.1.2.6 Line Points (LP)

3.1.2.6.1 Syntax

```
class LP                                // Line Point
{
}
```

3.1.2.6.2 Semantics

A **Line Point** indicates the location of the Line Segments (LSs), and Line Color Profiles (LCs) in the texture.

A Line Point is the meeting point between 2 adjacent Line Segments, i.e. every LS connects exactly 2 LPs. Every LC in the texture resides on a single corresponding LP.

In practice, the LP entity is not represented explicitly in the Texture data structures and bitstream, but it is mentioned in the primitive model for completeness. The LP location information can be derived from the LN's starting and ending Terminal Points and its Line Segments.

3.1.2.7 Terminal Point (TP)

3.1.2.7.1 Syntax

```
class TP                                // Terminal Point
{
    float x, y;

    BR  aBR[];                          // branches from this TP
    int  nBR;                           // number of branches from this TP
    int  junctionType;

    class BR
    {
        int    iLN;                     // index of LN that branches from this BR
        bool   bOut;                     // is this branch an "out-branch"?
        int    iLnType;                  // Line Type of the branching LN.
    }
}
```

3.1.2.7.2 Semantics

Terminal Points are a higher level structure, providing additional information regarding the coloring behavior at crossings, splittings and end points of lines. Every "Line" is terminated by exactly 2 TPs – one at each end.

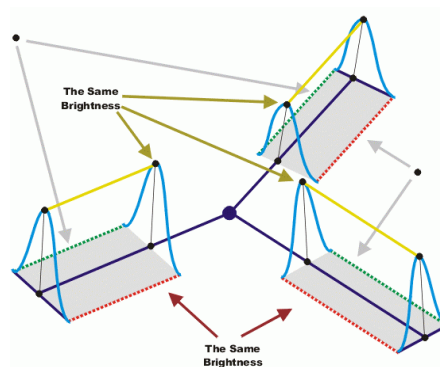


Figure 6 — Synthesized Texture Terminal Points (TP)

Terminal Points occur in the following circumstances:

- At any intersection in which 3 or more Lines are branching out – a TP is mandatory.
- At the end of a Line, where the Line does not connect to any other Line – a TP is mandatory.
- On a closed Line, (i.e. where the Line has no obvious end point), at least one TP must exist at an arbitrary point on the Line.
- At any point on a line, thus splitting that line into 2 "Lines".

Properties:

Name	Description
x, y	The coordinates of this TP in the Texture.
ABR	The array of branches from this TP.
NBR	The number of branches in aBR.
junctionType	The junction type of this TP: 0 - ALLRIDGE junction (separating). 1 - ALLSTRIPE junction (STRIPE = non-separating ridge). 2 - ALLEDGE junction (separating) 3 - SPLITTING junction (1 RIDGE in and 0 or 1 EDGES out)
class BR: a branch in which LNs go "in" or "out" of a TP.	
iLN	The index of the Line that branches from this BR.
bOut	Is this branch an "out-branch"? "out -branches" and "in -branches" are branches in which a Line "starts" and "ends" respectively. "in" and "out" is an arbitrary encoding decision, but every Line starts in an out-branch and ends in an in-branch.
iLnType	The line type of the branching Line. See LN.iLnType.

3.1.2.8 Line Color Profile (LC)

3.1.2.8.1 Syntax

```
class LC                                // Line Color Profiles
{
    float depth = 0;
}
```

```
class LCseparating                      // Separating (EDGE or RIDGE) Line Color Profile
{
    color colorFarLft, colorFarRgt;
    color colorMidLft, colorMidRgt;
}
```

```
class LCedge                            // EDGE Line Color Profile
extends LCseparating
{
    float width;
}
```

```
class LCridge                           // RIDGE Line Color Profile
extends LCseparating
{
    color colorCenter;
    float widthLft;
    float widthRgt;
}
```

```

class LCStripe                                     // STRIPE Line Color Profile
extends LC
{
    color colorCenter;
    color colorCurvature;
}

```

3.1.2.8.2 Semantics

Line Color Profiles (LCs) describe the coloring behavior along Lines.

Each colorXxx parameter describes the brightness of each of the 3 color components (Y, Cr, Cb) at a certain distance (center, middle, or far), and side (left or right) relative to the profile's geometrical center.

Line Color Profiles have 3 morphologies, corresponding to the 3 possible Line types: EDGE, RIDGE and STRIPE. All LCs on a Line are of the same type, given in the Line's iLnType.

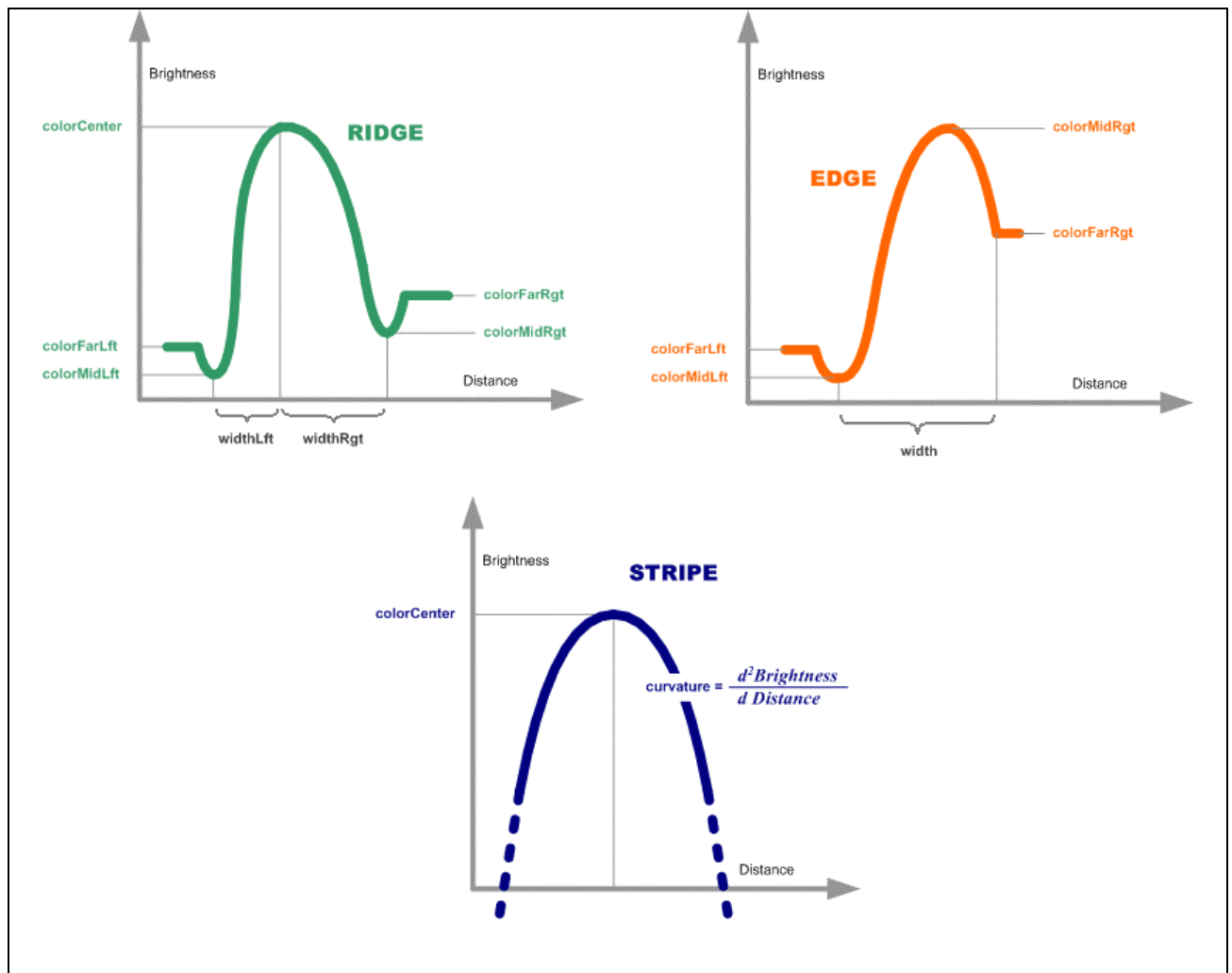


Figure 7 — Synthesized Texture Color Profiles (CP)

In the current implementation of SynthesizedTexture, an LC is provided for every location of a Line Point (LP) in the Texture, so its location is given by the corresponding LP's location²⁾.

A LN includes an array of LCs (aLC[nLC]) describing its coloring behavior, where nLC = nLS + 1. Thus for a Line with nLS segments, the location of LC_i for i=0..nLS-2 is the location (LS_i.x, LS_i.y), and the location of the last LC on the Line (i = nLS-1) is (TP_{LSi.iTpEnd.x}, TP_{LSi.iTpEnd.y}).

For convenience, the LC is also used to store the "depth" value for its respective Line Point.

Properties:

Name	Description
LC	Line Color Profile base class
Depth	The depth of this LC's Line at the location of this LC. Default=0.0.
LCseparating	Separating (EDGE or RIDGE) Line Color Profile base class
colorFarLft, colorFarRgt	The far-left and far-right colors of this LC.
colorMidLft, colorMidRgt	The mid-left and mid-right colors of this LC.
Lcridge	RIDGE Line Color Profile
ColorCenter	The central color of this LC.
widthLft, widthRgt	The left and right widths of this LC's Line at the location of this LC, in pixels.
Lcedge	EDGE Line Color Profile
Width	The width of this LC's Line at the location of this LC, in pixels.
Lcstripe	STRIPE Line Color Profile
ColorCenter	The central color of this LC.
ColorCurvature	The "color-curvature" of this stripe. If this stripe is described by a parabola $y=Ax^2+B$, then the color-curvature is the coefficient <i>A</i> .

3.1.2.9 Patch (PA)

3.1.2.9.1 Syntax

```
class PA
{
    float x, y;
    float r1, r2;
    float ang;

    color colorCenter;           // central color

    int iSL;

    float depth = 0;           // depth of this PA; may be absent.
}
```

2) This arrangement is economical as it saves the space needed to store LC location information, and does not come from deeper imaging grounds; note that LCs could theoretically be located on LNs according to an alternative scheme.

3.1.2.9.2 Semantics

A **Patch** is a small ellipse, typically a few pixels long, whose color is significantly different from its surrounding "area color".

The geometry of a Patch is described by the ellipse's center point, primary and secondary radius, and the angle of its primary radius relative to the texture's coordinate system.

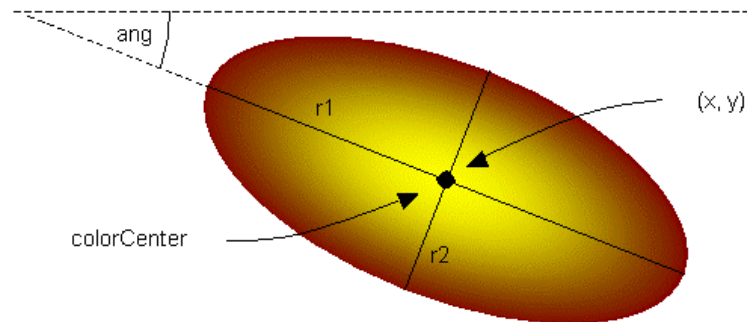


Figure 8 — Synthesized Texture Patch (PA)

The coloring of the Patch is described by the color at its geometric center, which can then be blended with the area color at its perimeter.

Properties:

Name	Description
x, y	The x and y coordinates of the geometrical center of the ellipse which represents this Patch.
R1, r2	The length of the primary and secondary radius respectively of the ellipse which represents this Patch, in pixels.
Ang	The angle of r1 relative to the Texture's X axis, in whole degrees, 0°..359°.
colorCenter	The central color of this PA.
ISL	The index of the sub-layer this PA is on.
Depth	The depth of this PA. Default=0.0.

3.1.2.10 Area Color Point (AC)

3.1.2.10.1 Syntax

```
class AC                                // Area Color Points
{
    float x, y;

    color colorCenter;    // central color

    float depth = 0;

    int iSL
}
```

3.1.2.10.2 Semantics

Area Color Points (ACs) describe the low-scale color changes in the areas between Lines, and more specifically between "separating" Lines, i.e. LNs of LineType EDGE and RIDGE.

Properties:

Name	Description
x, y	The coordinates of this AC in the Texture.
colorCenter	The color at the center of the AC.
depth	The depth of this AC. Default=0.0.
iSL	The index of the sub-layer this AC is on.

3.1.2.11 Layer (LY)

3.1.2.11.1 Syntax

```

class LY
{
    char    name[64];

    int     X0, X1, Y0, Y1;    // coords of LY's bounding rect
    bool    bHasDepthDeltas;

    SL      aSL[];
    int     nSL = 0;

    class SL
    {
        point3D orient;
        float    dist;
        bool     bOrthogonal;
        int      surfaceType;
        int      iLY;
    }
}

```

3.1.2.11.2 Semantics

Layers and **Sub-Layers** are auxiliary structures used in SynthesizedTexture authoring and encoding to group Texture primitives. Properties such as Depth and "controlling-polybone" are then attributed in bulk to all primitives in the Layer or Sub-Layer.

Properties:

Name	Description
name	The name of the layer, given by the author, used as reference in composition applications.
X0, X1, Y0, Y1	The coordinates of the layer's bounding rectangle.
bHasDepthDeltas	Do texture primitives in this layer have depth deltas (otherwise they inherit their depth from their position on the layer plane). Relevant only if decHeader.bHasDepthDeltas is true;
aSL[]	An array of sub-layers that are part of this LY.
nSL	Number of sub-layers in this LY.
class SL - sub-layer:	
orient	The normalized (x,y,z) representation of this sub-layer's orientation.
dist	The distance of this sub-layer in worldUnits. Typical 1..1000.
bOrthogonal	Is this sub-layer orthogonal to the line of sight? If true then orient is (0, 0, 1.0).
surfaceType	The type of geometry of this sub-layer. default: 0=PLANE.
iLY	Cross reference to the the index of the LY that this sub-layer is in.

3.1.2.12 Color

3.1.2.12.1 Syntax

```
class color
{
    int y, cr, cb;
    int R, G, B;
}
```

```
color::toRGB()
{
    R = y + 1.40200*cr
    G = y - 0.34414*cb - 0.71414*cr
    B = y + 1.77200*cb
}
```

3.1.2.12.2 Semantics

A **color** object can describe color in two methods: YCrCb and RGB.

YCrCb color values are used for encoding colors in the SynthesizedTexture bitstream, as they display better aggregation and compression behavior.

In the decoding process, YCrCb color values are converted to RGB colors, which are then used in Texture rendering.

3.1.3 Texture Rendering

3.1.3.1 General

The process of Texture Rendering reconstructs a displayable bitmap image from SynthesizedTexture primitives.

The reconstruction algorithm starts with the input parameters and computes the brightness (color) value of the synthesized image at each of its pixels. It consists of the following principal parts:

Computing brightness of Lines (Procedure BL below).

Computing brightness of Patches (Procedure BP below).

Computing Area coloring, i.e. brightness of the "Background", area between Lines and Patches (Procedure BB below).

Blending the computed brightness values into the final image (The MAIN Procedure below)

In the case of a color image these computations are performed independently for each color component.

3.1.3.2 Procedure MAIN: Computing Final Brightness Values

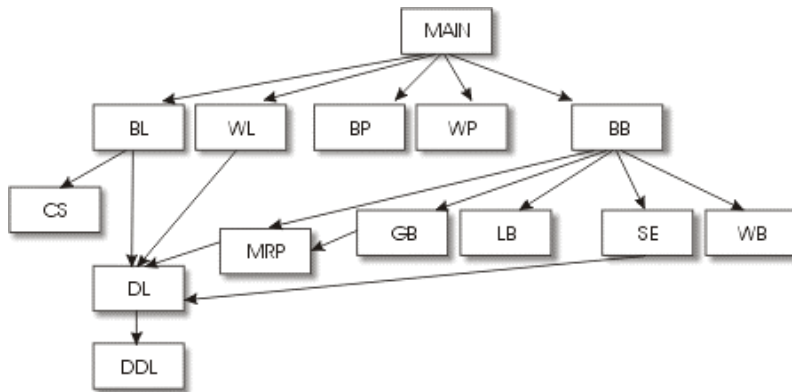


Figure 9 — Synthesized Texture Rendering Flow

For any point z in the image plane the final brightness $B(z)$ of the Synthesized image at the point z is computed according to the following formula:

$$(1) \quad B(z) = \frac{WL(z) \cdot BL(z) + WB(z) \cdot BB(z) + \sum_s WP_s(z) \cdot BP_s(z)}{WL(z) + WB(z) + \sum_s WP_s(z)}$$

Here $BB(z)$, $BL(z)$ and $BP_s(z)$ are the brightness functions of the Background, of the Lines and of the Patches, computed in the Procedures **BB**, **BL** and **BP**, respectively, and the sum $\sum_s$ runs over all the Patches P_s .

The weight functions $WL(z)$ and $WP_s(z)$ are computed in the Procedures **WL** and **WP**, respectively, and

$$WB(z) = 1 - \max(WL(z), WP_s(z)).$$

Division by the sum of all weight functions in formula (1) guarantees that their sum is identically 1 and that formula (1) is a normalized averaging.

3.1.3.3 Procedure BL: Brightness due to Lines

This procedure computes for any point z on the image the brightness $BL(z)$, contributed by the Line Color Profiles (LCs) of the Lines (LNs) in the texture.

$BL(z)$ needs to be computed only for those z which are "close enough" to at least one of the Lines in the texture, as expressed by the weight function WL .

Note: The most intuitive and natural way to define the brightness of a Line is to associate to it a coordinate system (uu, tt) , with $uu(z)$ the (signed) distance of the point z to the line along the normal direction, and $tt(z)$ the length parameter along the curve of the orthogonal projection $pp(z)$ of z onto the line.

Then the brightness cross-sections are computed according to the coordinate uu and interpolated along the line with respect to the coordinate tt .

The corresponding algorithm can be constructed. However, it provides some serious drawbacks:

- Actual computing of the coordinates uu and tt is a mathematically complicated task.

- Even for smooth curves without corners the normal direction is correctly defined only in a small neighborhood of the curve (of the size of a couple of pixels in realistic situations). Outside this neighborhood no natural mathematical solution exist for defining the normal, the projection etc.
- For spline curves with corners between some of their links (which usually appear in realistic situations) the normal is not defined even locally. Once more, the situation can be corrected by using the mid of the corner angles, but the global difficulties of (2) remain and algorithms become rather complex.

Consequently, we show below an algorithm, which can be considered as an approximation to the “ideal one” above. Its main advantage is that the “coordinates” uu and tt (called below u and t) can be computed independently for each Line Segment (link) of all the collection of Lines. Moreover, the computation can be ultimately rendered as rather efficient (although the description below may look somewhat complicated).

Below for any point z , $u(z)$ is the “distance of z to Lines”, $S(z)$ is the closest link to z (with respect to the distance u) in the collection of Lines, and $t(z)$ is the parameter, measuring the projection of z onto $S(z)$, rescaled to the segment $[0, 1]$. $S(z)$, $u(z)$ and $t(z)$ are computed by the Procedure **DL**, described below.

The Procedure **BL** distinguished whether the Line Segment $S(z)$ has a “free end” (i.e. an endpoint **TP**, not belonging to any other Line Segment) or not:

1. $S(z)$ does not have “free ends”.

Let C_1 and C_2 denote the equations of the two cross-sections (normalized to the unit width, as described in the Procedure **CS** below) at the two endpoints of the link $S(z)$. For $u(z) > 0$ let W_1 and W_2 denote the respective right widths RW_1 and RW_2 of the cross-sections at these points. For $u(z) < 0$ let W_1 and W_2 denote the respective left widths LW_1 and LW_2 of the cross-sections at these points. Then in each case

$$BL(z) = t(z) \cdot C_1\left(\frac{u(z)}{W(z)}\right) + (1 - t(z)) \cdot C_2\left(\frac{u(z)}{W(z)}\right)$$

where $W(z)$ is the interpolated width

$$W(z) = t(z) \cdot W_1 + (1 - t(z)) \cdot W_2$$

and the values $C_1\left(\frac{u(z)}{W(z)}\right)$ and $C_2\left(\frac{u(z)}{W(z)}\right)$ are computed by the procedure **CS**.

$S(z)$ has a “free end”.

If for this “free end” the parameter t is zero, the brightness $BL(z)$ is computed as above for $t(z) > 0$. For $-DE < t(z) < 0$,

$$BL(z) = [1 + t(z) / DE] \cdot C_1\left(\frac{u(z)}{W(z)}\right) - [t(z) / DE] \cdot BM,$$

and for $t(z) < -DE$,

$$BL(z) = BM.$$

Here DE is a positive tuning parameter, defining the shape of the end of a Line. BM is half of the sum of the brightness parameters $colorMidLft$ and $colorMidRgt$ of the cross-section at the free end.

If for this “free end” the parameter t is one, $t(z)$ is replaced by $1 - t(z)$ in the above formula.

Note: The formula above provides one of possible choices of the shape of Lines near their ends. It assumes that the cross-section brightness gradually descends to the “mid value” value BM inside the prescribed distance DE. Other shapes can be defined, by properly computing the width and the brightness in the neighborhood of the end point.

3.1.3.4 Procedure CS: Color Profiles (Cross-Sections) of Lines

This procedure computes a brightness value of an edge or a ridge (unit width) cross-section CS(u) for any given cross-section “interior” brightness parameters, as described above, and for any value of u.

In the Procedure BL u is the distance u(z) to the line, normalized by the width W(z) of the line, so the width parameter W is taken into account inside the BL, and it does not appear below. Similarly, the margin brightness parameters colorFarLft and colorFarRgt enter the computations in the Background brightness Procedure BB.

3.1.3.4.1 Edge Cross-Section.

Normalized edge cross-section NEC(u) is defined as follows (Figure 10 — Edge cross-section definition.):

$$NEC(u) = \begin{cases} u < -1, & 0 \\ u > 1, & 1 \\ -1 < u < 0, & 0.5(u+1)^2 \\ 0 < u < 1, & 1 - 0.5(u-1)^2 \end{cases}$$

Thus the recommended edge cross-section is composed of two symmetric parabolic segments.

For given brightness parameters colorMidLft and colorMidRgt, the value CS(u) is computed as

$$CS(u) = LB2 + (RB2 - LB2) \cdot NEC(u)$$

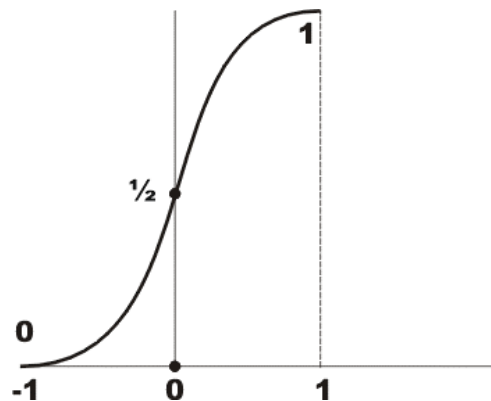


Figure 10 — Edge cross-section definition.

3.1.3.4.2 Ridge Cross-Section

As for edges, the width of the ridges is taken into account in the Procedure BL. Similarly, the margin brightness parameters colorFarLft and colorFarRgt enter the computations in the Background brightness Procedure BB. Consequently the ridge cross-section computed in the current Procedure CS, is the same for separating and non-separating ridges, and is defined by the parameters colorMidLft, colorCenter and colorMidRgt, as follows (Figure 11 — Ridge cross-section definition).

$$CS(u) = LB2 + (CB - LB2) \cdot NEC(2u + 1), \quad \text{for } u < 0, \text{ and}$$

$$CS(u) = RB2 + (CB - RB2) \cdot NEC(-2u + 1), \quad \text{for } u > 0.$$

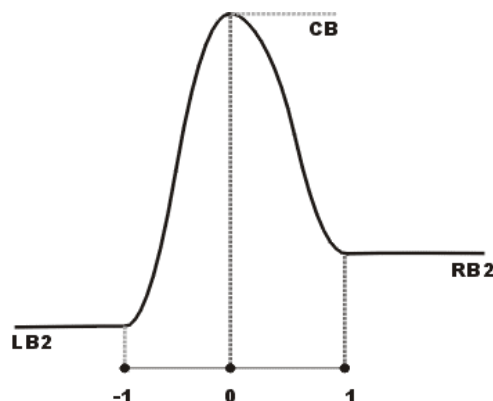


Figure 11 — Ridge cross-section definition.

Thus the recommended ridge cross-section is composed of two edge cross-sections, properly aggregated.

In the process of the blending of these cross-sections with the Background (which incorporates the margin brightness values *colorFarLft* and *colorFarRgt*) we get back essentially the same cross-section, as shown below (Figure 12 — Blending two cross-sections with the Background.).

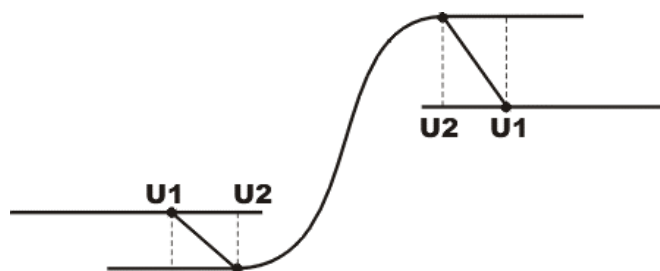


Figure 12 — Blending two cross-sections with the Background.

3.1.3.5 Procedure WL: Weight Function of Lines

This block computes the weight function $WL(z)$, which is used in a final blending of the Lines with the Background. The function $WL(z)$ is equal to one in a certain neighborhood of the Lines, and is zero outside of a certain larger neighborhood.

More accurately:

$$WL(z) = \begin{cases} 1 & |u(z)| < UL_2 \cdot W(z) \\ \frac{W(z) \cdot UL_1 - |u(z)|}{W(z) \cdot (UL_1 - UL_2)} & UL_2 \cdot W(z) < |u(z)| < UL_1 \cdot W(z) \\ 0 & |u(z)| > UL_1 \cdot W(z) \end{cases}$$

The distance $u(z)$ is computed in the Procedure **DL**.

UL_1 and UL_2 are tuning parameters; see the last section “Tuning Parameters”.

Figure 13 shows a typical cross-section and a general shape of the weight function $WL(z)$.

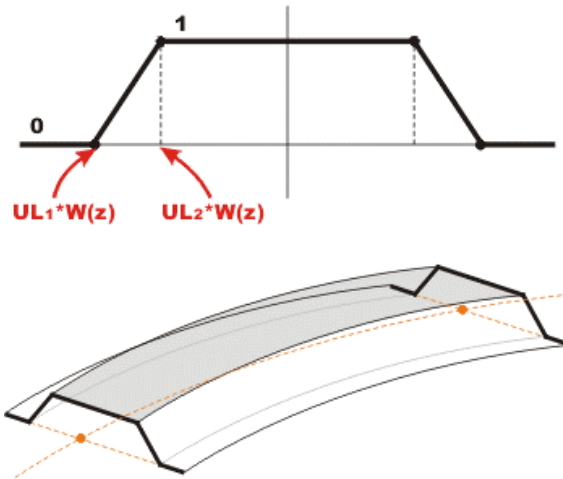


Figure 13 — Typical cross-section and general shape of $WL(z)$

3.1.3.6 Procedure DL: Distance to Lines

This Procedure computes for any point z in the texture:

1. The point $p(z)$ on the Lines which is nearest to z , i.e. the “projection” $p(z)$ of z onto the set of Lines.
- d) The distance $u(z)$ between z and $p(z)$.
- e) The link $S(z)$ on which $p(z)$ resides.
- f) The proportion $t(z)$ in which $p(z)$ divides the link $S(z)$.

Note $u(z)$, $p(z)$ and $t(z)$ are NOT exactly the Euclidean distance, the corresponding mathematical projection and proportion respectively; however, in most cases they give a reasonable approximation for these mathematical entities.

These data are computed in the following steps:

2. For each link (Line Segment, LS) S_i in the texture, the corresponding $p_i(z)$, $u_i(z)$, $t_i(z)$ are computed in Procedure DDL (See the Synthesized-C figure below)
 - g) $S(z)$ is defined as the link S_j , for which the minimum of the absolute values $|u_i(z)|$ is attained (See the figure Synthesized-D below)
 - h) $u(z)$ is defined as the function $u_j(z)$ for the link $S_j = S(z)$
 - i) $t(z)$ is defined as $t_j(z)$ for the above link S_j

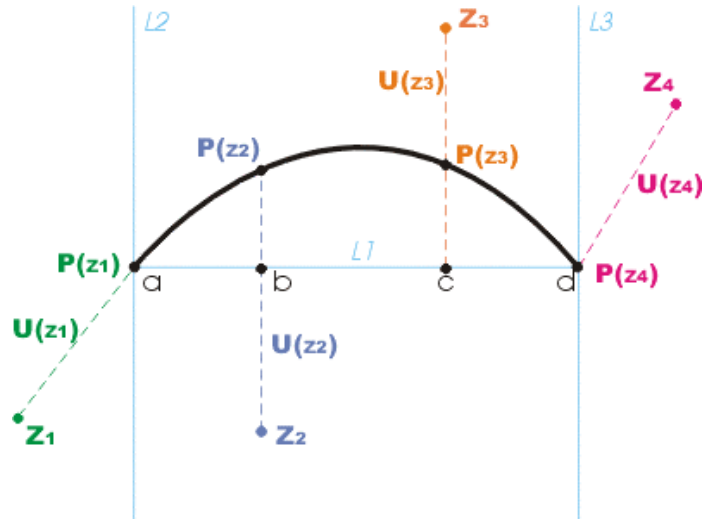


Figure 14 — Synthesized-C

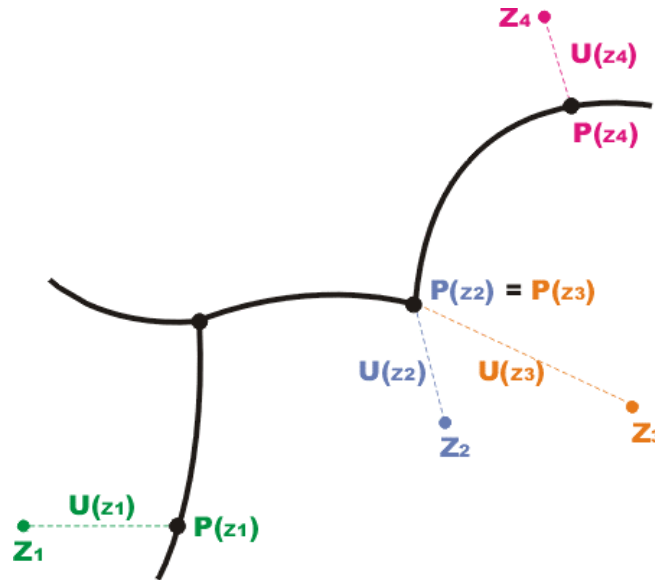


Figure 15 — Synthesized-D

3.1.3.7 Procedure DDL: Distance to a Line Segment

This procedure computes for any point z its (signed) distance $u(z)$ to a given link S (Line Segment, LS), the projection $p(z)$ of the point z onto the link S and the parameter $t(z)$. The Procedure is essentially represented on figure Synthesized-C above (which shows, in particular, equidistant lines for the points z_1 and z_4).

The straight oriented segment $[a, d]$, joining the end points of the link S is constructed, with the orientation, induced from the orientation of the Line, containing S . l_1 is the straight line, containing the segment $[a, d]$. l_2 and l_3 are the straight lines, orthogonal to l_1 and passing through a and d , respectively.

Now, for any z in the image plane, the function $u(z)$ is constructed as follows:

For z between l_2 and l_3 , the absolute value $|u(z)|$ is the length of the segment, joining z and S and orthogonal to l_1 .

For z left to l_2 , $|u(z)|$ is defined as follows:

$$|u(z)| = [d(z, l_1)^2 + Dd(z, l_2)^2]^{1/2}.$$

Here $d(z, l_1)$ and $d(z, l_2)$ are the distances from z to l_1 and

l_2 respectively. D is a tuning parameter, with a typical value $D = 4$.

For z right to l_3 , $|u(z)|$ is defined as

$$|u(z)| = [d(z, l_1)^2 + Dd(z, l_3)^2]^{1/2}.$$

Let ll be an oriented line, formed by the interval of the line l_1 from infinity to a , then by the link S from a to d , and then by the interval of the line l_1 from d to infinity.

For z right to ll (with an orientation as above) the sign of $u(z)$ is “+”. For z left to ll , the sign of $u(z)$ is “-”.

For z between l_2 and l_3 , the projection $p(z)$ is the intersection point of S and of the segment, joining z and S and orthogonal to l_1 . For z left to l_2 , $p(z)$ is a , and for z right to l_3 , $p(z)$ is d .

For any z , $t(z)$ is the proportion, in which the projection of z onto the line l_1 subdivides the segment $[a, d]$. For example, for the point z_2 and z_3 on Fig Synthesized-D above. $t(z_2) = (b-a)/(d-a)$, and $t(z_3) = (c-a)/(d-a)$, respectively. For z left to l_2 , $t(z) < 0$, and for z right to l_3 , $t(z) > 1$.

Note: The special form of the function $u(z)$ above (for z outside the strip between l_2 and l_3) is motivated by the following reason: when computing in the Procedure BL the brightness of the line near a sharp corner, the form of the distance function $u(z)$ determines which link will be taken as the closest to the points in the sector stressed on the Figure below. For the distance, computed as above, with the parameter $D > 1$, this choice is matched with the sign of $u(z)$, as defined above. If we would have chosen $D < 1$, for z in the sector stressed on the figure below, the choice of the nearest link, together with the proposed computation of the sign of $u(z)$, would produce a color from the incorrect side of the line. See the figure below.

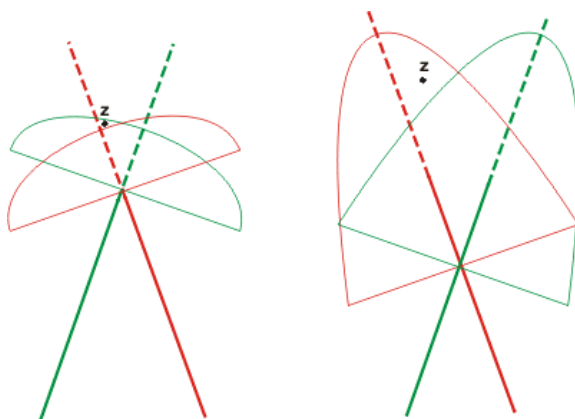


Figure 16 — Distance based selection of line segment

3.1.3.8 Procedure BP: Brightness of Patches

Let C_x , C_y , R_1 , R_2 , a , colorCenter and MB be the parameters of a certain Patch P_s , as described above.

Let M be the linear transformation of the plane, transforming the basis ellipse of the Patch to the unit circle. M is a product of the translation by $(-C_x, -C_y)$, the rotation matrix to the angle $-a$, and the rescaling $1/R_1$ and $1/R_2$ times along the x and y axes, respectively. If we put for

$$z = (x, y), (x'(z), y'(z)) = M(x, y) = M(z),$$

then the equation of the basis ellipse of the Patch is given by

$$x'(z)^2 + y'(z)^2 = 1.$$

The brightness function $BP_s(z)$ of the Patch is then given by

$$BP_s(z) = 0 \text{ for } x'(z)^2 + y'(z)^2 > UP1^2,$$

$$BP_s(z) = MB \text{ for } 1 < x'(z)^2 + y'(z)^2 < UP1^2, \text{ and}$$

$$BP_s(z) = MB + (CB - MB) \cdot (1 - x'(z)^2 - y'(z)^2), \text{ for } x'(z)^2 + y'(z)^2 < 1.$$

Here $UP1 > 1$ is a parameter. See the Figure below.

3.1.3.9 Procedure WP: Weight Function of Patches

The weight function $WP_s(z)$ for a Patch P_s as above is defined by

$$WP_s(z) = 0 \text{ for } uu(z) > UP1,$$

$$WP_s(z) = 1 \text{ for } uu(z) < UP2, \text{ and}$$

$$WP_s(z) = (UP1 - uu(z)) / (UP1 - UP2)$$

for $uu(z)$ between $UP2$ and $UP1$,

where $uu(z)$ denotes the square root of $x'(z)^2 + y'(z)^2$.

Here $UP2$, $1 < UP2 < UP1$, is another tuning parameter. See the figure below.

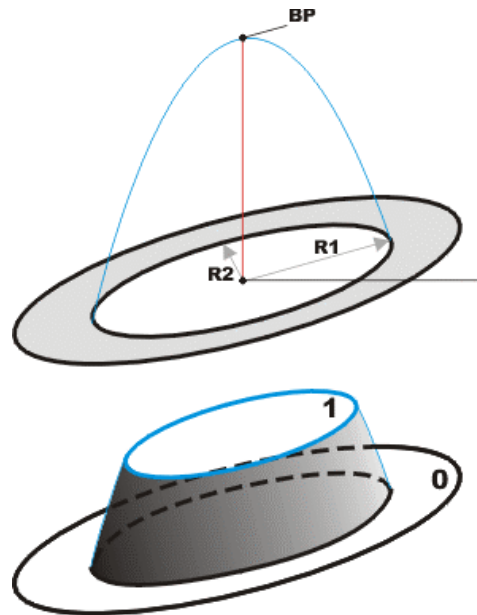


Figure 17 — Synthesized Texture Patch and its weight function

3.1.3.10 Procedure BB: Brightness of Background

This Procedure computes the brightness value of the Background at any point z of the image. This value is obtained as a result of interpolation between the “global” Background brightness values, the margin brightness values of the Lines and the brightness values at the Area Color Points (ACs). The main difficulty is that the interpolation is not allowed to cross the separating lines. To overcome this difficulty a special “distance” d between the points on the image is introduced, which is the length of the shortest pass, joining these points, and not crossing separating Lines, and which is computed in the Procedure **SE** below. Then averaging weights are computed through the distance d .

This block uses as an input a certain collection of the ACs Z_i , (containing the input ACs, as described above, and the margin representing points, produced by the block “**MRP**”, described below). At each point Z_i the brightness value Bb_i is given.

The Background brightness value $BB(z)$ is finally produces by the block BB as follows:

$BB(z)$ is the weighted sum of the global brightness GB and of the Local brightness functions $Bb_i(z)$ over all the ACs Z_i :

$$(2) \quad BB(z) = (1/S_1(z))[WG(z)BG + \sum_i WR(d(z, Z_i))Bb_i(z)]$$

$$\text{Here } S_1(z) = WG(z) + \sum_i WR(d(z, Z_i)),$$

so the expression (2) is normalized to provide a true averaging of the corresponding partial values.

The global brightness value GB is provided by the Procedure **GB** below, or by any of the Gradient Nodes.

The computation of the Local brightness functions $Bb_i(z)$ is performed in the Procedure **LB** below.

The distance functions $d(z, Z_i)$ are computed in the Procedure **SE** below.

The computation of the weight functions $WR(d(z, Z_i))$ is performed in the Procedure **WB** below.

The weight $GW(z)$ of the global Background value GB is defined as

$$GW(z) = 1 - \max_i WR(d(z, Z_i)).$$

$GW(z)$ is zero at any z , where at least one of the weights of the representing points is 1, and $GW(z)$ is one at any z where all the weights of the ACs vanish.

3.1.3.11 Procedure GB: Global Brightness of Background

This Procedure computes the global Background value GB , which appears in the expression (2) in the Procedure BB.

By definition, if the point z is inside the Background region of a Sub-Texture number r , for which the global value GB_r is defined, GB is equal to this global value GB_r . If the point z is inside the Background region of a Sub-Texture, for which the global Background value is not defined, GB is equal to the default global value DGB . If DGB is not defined, GB is equal to zero. Alternatively, Color Gradients can be used.

The current procedure consists in a signal expansion that transmits to each pixel its Sub-Texture number. We describe it shortly, since it essentially belongs to a higher data representation level.

First the procedure MRP is applied, which creates margin representing points, carrying the corresponding Sub-Texture numbers. These numbers are taken from the corresponding poly-links.