Interactive Real-Time Three-Dimensional Visualisation of Virtual Textiles

Michael Stuart Alexander Robb, B.Sc, B.Sc. (Hons), M.Sc.

Thesis submitted

for the

Degree of Doctor of Philosophy

Heriot-Watt University

Department of Computer Science

December 2009

Abstract

Virtual textile databases provide a cost-efficient alternative to the use of existing hardcover sample catalogues. By taking advantage of the high performance features offered by the latest generation of *programmable graphics accelerator boards*, it is possible to combine *photometric stereo* methods with *3D visualisation* methods to implement a virtual textile database. In this thesis, we investigate and combine *rotation invariant texture retrieval* with interactive visualisation techniques.

We use a *3D surface representation* that is a generic data representation that allows us to combine *real-time* interactive *3D visualisation* methods with present day *texture retrieval* methods. We begin by investigating the most suitable data format for the *3D surface representation* and identify *relief-mapping* combined with Bézier surfaces as the most suitable *3D surface representations* for our needs, and go on to describe how these representation can be combined for *real-time rendering*. We then investigate ten different methods of implementing *rotation invariant texture retrieval* using *feature vectors*. These results show that first order statistics in the form of histogram data are very effective for discriminating colour *albedo* information, while *rotation invariant gradient maps* are effective for distinguishing between different types of *micro-geometry* using either first or second order statistics.

Dedication

To Mum, David, Marie-Louise, Dad and
Fluffy, Tiga, Genjie and Julie and Snooks

Acknowledgements

The research described by this thesis was conducted in the Department of Mathematics and Computer Science at Heriot-Watt University in Edinburgh, Scotland. The location for this research was in TextureLab at the Riccarton Campus as part of the Virtex project. The journey of discovery for this thesis has taken many years of work in the fields of *real-time* computer graphics and signal processing.

First and foremost, I wish to thank my supervisor, Professor Mike Chantler for offering me the opportunity to conduct this research, his continual support and taking the time to review the many chapters of this thesis. I would also like to thank Dr. Patrick Green for his advice on the design of psychophysical experiments. I would also like to thank Mark Timmins from the School of Textiles and Design for his helpful advice during the early years of this thesis. I would especially like to thank Andy Spence for helping to co-author the research papers that we have published over the years. I would also like to thank the team at TextureLab for their many valuable contributions: Ged McGunnigle, Jerry Wu, Christine Gullón, Ondřej Drbohlav, Jiri Filip, Fraser Halley, Pratik Shah, Junyu Dong, Khemraj Emrith and Stephano Padilla. I would also like to thank Dr. J.P Siebert for taking the time to act as the external examiner for this thesis.

I would also like to thank the anonymous reviewers of the papers that were submitted for publication. I would also like to thank the ESPRC for providing the research funding for the Virtex project. Finally, I would like to thank my family for their support during the many years that this thesis has taken from start to completion.

ACADEMIC REGISTRY
Research Thesis Submission

Name*:*	Mr. Michael Robb		
School/PGI:	School of Mathematics and Computer Science		
Version: *(i.e. First, Resubmission, Final)*		Degree Sought:	Ph.D.

Declaration

In accordance with the appropriate regulations I hereby submit my thesis and I declare that:

1) the thesis embodies the results of my own work and has been composed by myself
2) where appropriate, I have made acknowledgement of the work of others and have made reference to work carried out in collaboration with other persons
3) the thesis is the correct version of the thesis for submission*.
4) my thesis for the award referred to, deposited in the Heriot-Watt University Library, should be made available for loan or photocopying, subject to such conditions as the Librarian may require
5) I understand that as a student of the University I am required to abide by the Regulations of the University and to conform to its discipline.

* *Please note that it is the responsibility of the candidate to ensure that the correct version of the thesis is submitted.*

Signature of Candidate*:*		Date:	

Submission

Submitted By *(name in capitals):*	
Signature of Individual Submitting:	
Date Submitted:	

For Completion in Academic Registry

Received in the Academic Registry by *(name in capitals):*			
Method of Submission *(Handed in to Academic Registry; posted through internal/external mail):*			
Signature:		Date:	

Table of Contents

List of Figures

List of Tables

Principal Symbols

Symbol	Meaning	Section first introduced	Type
$\mathbf{l}$	Illumation vector	4.1.1	vector
rm_ray	The relief-mapping surface ray intersection function	E.1.10	function
$M_{projection}$	The camera perspective projection matrix.	E.1	matrix
M_{albedo}	The texture coordinate transformation matrix for the albedo and normalmap textures.	E.1	matrix
M_{shadow}	The tranformation matrix for the shadow map texture	E.1	matrix
$M_{projector}$	The transformation matrix for the projection texture.	E.1	matrix
$M_{modelview}$	The combined geometry and camera transformation matrices.	E.1	matrix
$\mathbf{gl\_texcoord_n}$	OpenGL output texture coordinates	E.1.1	vector
gl_multitexcoor	OpenGL input texture coordinates	E.1.1	vector
$\mathbf{c_{uv}}$	The texture coordinate vector	E.1.10	vector
$k_{dodiffuse}$	The control flag for enabling diffuse lighting	E.2	scalar
$k_{dospecular}$	The control flag for enabling specular lighting	E.2	scalar
$k_{doreliefmapping}$	The control flag for relief-mapping,	E.2	scalar
$k_{doshadow}$	The control flag for shadow mapping,	E.2	scalar
k_{shadow}	The shadow mapping depth texture,	E.2	scalar
$\mathbf{k_{selfshadow}}$	A factor to represent self-shadowing,	E.2	vector
$\mathbf{n_{normalmap}}$	The surface normal from a normalmap texture	E.1.11	vector
$d_{diffuse}$	The dot product for the diffuse term.	E.1.13	scalar
$\mathbf{c_{zero}}$	A constant representing no contribution.	E.1.14	vector
$\mathbf{l_r}$	The reflected light vector in tangent space.	E.1.15	vector
$\mathbf{d_{specular}}$	The dot product of the specular lighting term.	E.1.16	scalar
$\mathbf{c_{specfactor}}$	The specular lighting contribution.	E.1.17	vector

Symbol	Meaning	Section first introduced	Type
$\mathbf{c}_{\mathbf{diffuse}}$	The colour from the diffuse term.	E.1.18	vector
$\mathbf{c}_{\mathbf{albedo}}$	The colour from the albedo texture.	E.1.18	vector
$\mathbf{l}_{\mathbf{diffuse}}$	The diffuse contribution of the light source.	E.1.18	vector
$\mathbf{l}_{\mathbf{specular}}$	The specular contribution of the light source.	E.1.19	vector
$\mathbf{c}_{\mathbf{specular}}$	The resulting specular contribution of the light source.	E.1.19	vector
gl_vertex	OpenGL input vertex position	E.1.2	vector
$\mathbf{c}_{\mathbf{zero}}$	A constant colour representing no contribution.	E.1.20	vector
$\mathbf{l}_{\mathbf{ambient}}$	The ambient contribution of the light source.	E.1.21	vector
$\mathbf{c}_{\mathbf{albedo}}$	The colour retrieved from the albedo texture,	E.1.21	vector
$\mathbf{c}_{\mathbf{ambient}}$	The contribution from the ambient term.	E.1.21	vector
$\mathbf{c}_{\mathbf{final}}$	The final colour for the pixel fragment,	E.1.22	vector
$\mathbf{c}_{\mathbf{one}}$	A constant representing maximum colour intensity.	E.1.25	vector
gl_position	OpenGL output vertex position	E.1.3	vector
$\mathbf{e}_{\mathbf{tangentspace}}$	The eye vector transformed into tangent space.	E.1.5	vector
$\mathbf{l}_{\mathbf{position}}$	The position of light source in world space.	E.1.6	vector
$\mathbf{l}_{\mathbf{tangentspace}}$	The light vector transformed into tangent space.	E.1.6	vector
$\mathbf{l}_{\mathbf{direction}}$	The direction of the light source in world space.	E.1.7	vector
$\mathbf{l'}_{\mathbf{tangentspace}}$	The normalized light vector in tangent space.	E.1.8	vector
$\mathbf{e'}_{\mathbf{tangentspace}}$	The normalized eye vector in tangent space.	E.1.9	vector
tp	True positive	2.4.2	scalar
fp	False positive	2.4.2	scalar
fn	False negative	2.4.2	scalar
tn	True negative	2.4.2	scalar
(l_x, l_y, l_z)	Individual elements of the illumination vector	4.1.1	vector
p	First gradient field for photometric stereo	4.1.9	scalar
q	Second gradient field for photometric stereo	4.1.9	scalar
τ	Tilt angle	4.1.1	scalar

Symbol	Meaning	Section first introduced	Type
σ	Slant angle	4.1.1	scalar
ϕ_p	Gradient angle in radians	4.2	scalar
ϕ_q	Gradient angle in radians	4.2	scalar
p'	Tangent vector in tangent space	4.2	vector
q'	Binormal vector in tangent space	4.2	vector
$\mathbf{l}_1, \mathbf{l}_2, \mathbf{l}_3$	Direction vectors of incident illumination	4.1.2	vector
L	Matrix formed from direction vectors $\mathbf{l}_1, \mathbf{l}_2, \mathbf{l}_3$	4.1.3	matrix
n	Unit surface normal	4.1.4	vector
k_ρ	Reflectance factor (albedo)	4.1.5	scalar
t	Scaled surface normal	4.1.6	vector
(t_x, t_y, t_z)	Components of the scaled surface normal	4.1.7	scalar
$M_{tangentspace}$	The inverse transpose of $M_{eyespace}$	5.5.2	matrix
t	Abbreviated form of the tangent vector	5.5.2	vector
b	Abbreviated form of the binormal vector	5.5.2	vector
n	Abbreviated form of the normal vector	5.5.2	vector
p	Abbreviated form of the world space coordinate	5.5.2	vector
(t_x, t_y, t_z)	Scalar components of the tangent vector	5.5.2	vector
(b_x, b_y, b_z)	Scalar components of the bi-normal vector	5.5.2	vector
(n_x, n_y, n_z)	Scalar components of the surface normal vector	5.5.2	vector
(p_x, p_y, p_z)	Scalar components of the point on the surface	5.5.2	scalar
p	Geometric point	5.5.3	vector
U	Row vector	5.5.3	matrix
M_{basis}	Basis matrix	5.5.3	matrix
C	The set of control points	5.5.3	matrix
$F(u)$	The Bézier curve	5.5.3	function
n	The degree of the curve	5.5.3	scalar
c_i	The control points for the Bézier curve	5.5.3	scalar
c_n	The control points for a spline curve	5.5.3	scalar
$p(u)$	The cubic Bézier spline curve	5.5.3	function

Symbol	Meaning	Section first introduced	Type
$p'(u)$	The first derivative of the spline curve	5.5.3	function
b_{nm}	The coefficients of the basis matrix	5.5.3	scalar
n	The degree of the Bézier rectangle in u	5.5.3	scalar
m	The degree of the Bézier rectangle in v	5.5.3	scalar
$F(u,v)$	The Bézier rectangle function	5.5.3	function
c_{ij}	The control points for the Bézier rectangle	5.5.3	scalar
$F(u,v,w)$	The Bézier triangle function	5.5.3	function
(u,v,w)	Parametric coordinates in three dimensions	5.5.3	scalar
c_{ijk}	The control points for the Bézier triangle	5.5.3	scalar
$\mathbf{l_i}$	Resulting output intensity for lighting calculations	6.1.4	vector
$\mathbf{k_a}$	The fraction of light emitted by ambient reflection	6.1.4	vector
$\mathbf{k_d}$	The fraction of light emitted by diffuse reflection	6.1.4	vector
$\mathbf{k_s}$	The fraction of light emitted by specular reflection	6.1.4	vector
$\mathbf{k_{sp}}$	Specular power factor of a material	6.1.4	vector
$\mathbf{f_{att}}$	The fraction of light that reaches the surface due to the attenuation of light.	6.1.4	vector
$\mathbf{e}$	The eye vector (normalized direction vector pointing towards viewpoint)	6.1.4	vector
$\mathbf{l}$	The light vector (normalized direction vector pointing towards the light source)	6.1.4	vector
$\mathbf{r}$	The light vector reflected through the surface normal	6.1.4	vector
$f(x,y)$	A 2D signal in the spatial domain	7.6.1	function
$F(u,v)$	A 2D signal in the frequency domain	7.6.1	function
(u,v)	Coordinates in the frequency domain	7.6.1	scalar
(x,y)	Coordinates in the spatial domain	7.6.1	scalar
(r,s)	Dimensions of the frequency domain	7.6.1	scalar
$F_r(r,\sigma_r,r_{centre})$	The Ring filter function	7.6.3	function

Symbol	Meaning	Section first introduced	Type
r	Radius in frequency space	7.6.3	scalar
σ_r	Standard deviation of radius in frequency space	7.6.3	scalar
r_{centre}	Arithmetic mean of radius in frequency space	7.6.3	scalar
$F_w(\theta,\sigma_{\theta,}\theta_{centre})$	The Wedge filter function	7.6.3	function
θ	Angle in the frequency domain	7.6.3	scalar
σ_θ	Standard deviation of angle in the frequency domain	7.6.3	scalar
θ_{centre}	Arithmetic mean of angle in the frequency domain	7.6.3	scalar
$I_f(u,v)$	Image of the filter in the spatial domain	7.6.3	array
$I_t(u,v)$	Image of the texture in the spatial domain	7.6.3	array
$I'_f(u,v)$	Image of the filter in the frequency domain	7.6.3	array
$I'_t(u,v)$	Image of the texture in the spatial domain	7.6.3	array
$r(u,v)$	Combined texture and filter image in the frequency domain	7.6.3	array
$e(x, y)$	Combined texture and filter image in the spatial domain	7.6.3	array
O_n	Output value of the filter	7.6.3	scalar
$F_g(r,\sigma_r,$ $r_{centre},\theta,\sigma_\theta,\theta_{centre})$	The Gabor filter function	7.6.5	function
$F_s(r,\sigma,\tau)$	The Schmid filter function	7.6.6	function
$F_0(\sigma,\tau)$	A constant added to Schmid filters to ensure a zero DC component.	7.6.6	function
$F_{gm}(\sigma,r)$	The Gaussian filter function	7.6.6	function
$F_{LoG}(\sigma,r)$	The Laplacian-of-Gaussian filter function	7.6.6	function
$G'(x, y,\theta)$	The Gaussian first derivative filter function	7.6.6	function
$G''(x, y,\theta)$	The Gaussian second derivative filter function	7.6.6	function

Abbreviations

Abbreviation	Meaning	Section first introduced
2D	Two-Dimensional	2.3.11
3D	Three-Dimensional	2.4.3
BRDF	Bidirectional Reflectance Distribution Function	2.1.1
BSDF	Bidirectional Scattering Distribution Function	2.1.3
BSSDF	Bidirectional Subsurface Scattering Distribution Function	2.1.1
BSSRDF	Bidirectional Scattering Surface Reflectance Distribution Function	2.1.1
BTDF	Bidirectional Transmission Distribution Function	2.1.3
BTF	Bidirectional Texture Function	2.1.5
CPU	Central Processing Unit	2.1.1
CUReT	Columbia-Utrecht Reflectance and Texture Database	3.3
FFT	Fast Fourier Transform	3.3
GPU	Graphics Processing Unit	2.2.3
IFFT	Inverse Fast Fourier Transform	7.6
JPEG	Joint Photographics Experts Group	2.3.1
KBytes	Kilobytes	2.3.1
LRGB	Luminance, Red, Green, Blue	2.1.7
MBytes	Megabytes	2.2.8
MR-4	Maximum Response 4	2.3.19
MR-8	Maximum Response 8	2.3.19
NURBS	Non-Uniform Rational B-Spline	2.1.1
PTM	Polynomial Texture-mapping	2.1.7
RGB	Red, Green and Blue	2.1.5
RGBA	Red, Green, Blue and Alpha	4.2
VDM	View-Dependent Displacement Mapping	5.4.5

Definition of Terms

Terms	Definition	Section First introduced
3D Geometric model	A data structure used to represent a 3D object. It may be comprised from a collection of vertices and polygons or a collection of parametric surfaces.	1.1
3D graphics pipeline	A series of stages used to render 3D views given a set of geometric data.	5.4
3D surface representation	The representation of 3D characteristics of a surface. This includes gradient and heightmap information along with colour information.	1.1
3D visualisation	The presentation of mathematical data in a visual form.	1.2
Accuracy	The ability of an information retrieval system to select relevant items and discard non relevant items.	2.2.6
Accuracy rate	The ability of an information retrieval system to select relevant items and discard non relevant items.	2.2.6
Albedo	The basic colour information of a texture sample under ambient light conditions.	1.1
Alpha channel	A fourth channel of data used to augment the existing red, green and blue colour channels of a pixel. Used for a variety of purposes including transparency, heightmap and specularity.	2.1.8
Basis matrix	A matrix used to define high-order curves and surfaces such as NURBS.	5.5.3
Bisection method	A method of root finding within an interval. The interval is subdivided in half until the root is found.	6.1.3
Boundary histogram	A colour histogram in which the data is corrected for variations in illumation conditions.	2.3.3
Camera projection space	The area of a scene visible to a camera perspectively projected into the cube (-1,-1,-1) to (1,1,1).	5.5.2
Camera space	The local coordinate system of a camera.	5.5
Chromatic aberration	The rainbow effect due to the fact that refraction angle of a single photon of light is proportional to the wavelength.	6.1.5
Colour histogram	A method of texture classification based only on the frequency distribution of individual colours within the texture.	2.3.3

Terms	Definition	Section First introduced
Colour ratio histogram	A histogram in which the derivative or the logarithm of the data is used to construct the table of values	2.3.3
Combined filter banks	A method of texture classification that combines together all of the existing filter methods.	7.4.4
Control net	A series of vertices used to control the shape of a parametric surface.	5.5.3
Control points	A series of vertices used to control the shape of a parametric surface.	2.5
Curvature	The level at which a two-dimensional parameteric surface curves in three dimensional space. As there are two parametric coordinates, a two-dimensional can either have a curvature level of 0 (planar), 1 (cylindrical) or 2 (torus, torus-knot, helix)	5.4.5
DC component	A constant used by the Schmid filter set in order to make the average sum to zero.	2.3.17
Depth texture map	Another term for a heightmap. The depth texture map represents the depth into the geometric model.	6.1.4
Displacement map	A texture map in which each pixel represents the displacement on the resulting surface.	1.1
Environment mapping	A method of giving a material a reflective appearance. The reflection vector for each pixel fragment is calculated, and then used to index a texture map (the environment map) which can be a variety of shapes (two hemispheres, a cylinder or a cube).	6.1.5
Eye vector	The direction vector between a point on the surface of a 3D geometric model and the observer.	5.4
Fallout	The ability of an informationr retrieval system to retrieve non-relevant items to the search query.	2.4.2
Feature vector	A compact set of data values that represent the unique properties of a database entry. These may be easily compared to facilitate texture retrieval.	1.3
Filter bank	A series of filters that operate in the frequency domain, with the resulting outputs used to create a feature vector.	2.3.14
Fine scale surface detail	The appearance of very small detail such as wrinkles, dents and bulges on a surface, which change as the position of the light source changes.	2.1.5

Terms	Definition	Section First introduced
Fragment shader	A programmable stage in the graphics pipeline.	5..4.7
Frequency domain	The resulting image created by applying the FFT to a standard texture image. The resulting image represents the energy contribution of particular frequencies and directions.	2.3
Fresnel reflection	The effect that the sides of an object viewed from side on will appear more reflective than those viewed from straight on.	6.1.4
Gabor filters	A series of frequency domain filters which are sensitive to both direction and frequency.	2.3.16
Gaussian filter	A frequency domain filter used to implement the smoothing of an image.	2.3.18
General lighting function	A mathematical equation which models the appearance of a surface down to the points and times of absorption and emissson of individual photons of light.	2.1.2
Geometric point	A single point on the surface of a 3D geometric model.	2.1.10
Gloss maps	A texture maps that represents the reflective properties of a material	2.1.8
Gradient field	A texture image that defines the gradient in a selected axis of an image.	1.1
Gradient histogram	A histogram in which each bin represents a particular range of gradient values	7.8.3.1
Heightmap	A texture image that defines the height of each pixel on the texture.	2.1.11
Histogram	A statistical method of analyzing data in which data elements are grouped together according to particular ranges.	7.8.3.1
Horizon mapping	A 3D surface representation method which defines a horizon map for each texture pixel. The horizon map uses a single bit to indicate whether the light source is visible from a particular direction.	5.4.6
Illumination vector	The direction vector towards the light source.	4.1
Image data	The pixel information contained within a two-dimensional array of pixels.	1.1
Integration methods	A method of converting surface normal information into a heightmap.	1.1

Terms	Definition	Section First introduced
Inter-reflectance	The ability of a lighting equation to model light reflected between different areas of the same surface.	3.3
Interval analysis	A method of root-finding with complex surfaces.	5.5.3
Laplacian-of-Gaussian	A filter in the frequency domain measuring the 2^{nd} spatial derivative of an image.	2.3.19
Leung-Malik filter bank	A frequency domain filter bank based on a collection of Gaussian, Laplacian-of-Gaussian, Edge and bar filters.	2.3.18
Light projection space	The local coordinate system of the projective texture of a light source.	6.1.5
Light projection texture	The texture of a light source that is used to model the light distribution of that light source.	5.5.2
Light vector	The direction vector between a point on the surface of a 3D geometric model and a light source.	5.5.4
Lighting equation	A mathematical equation used to calculate the resulting colour of a point on a surface.	2.1.6
Lighting model	A mathematical equation used to calculate the resulting colours.	1.2
Linear searching	The first stage in interval analysis. This involves stepping forward along the direction vector of the ray at fixed intervals.	5.4.9
Local coordinate system	The coordinate system for a particular entity in a 3D scene.	5.5.2
Macro-texture	The basic topology that gives a 3D object its recognizable shape. This is the difference in shape in objects such as a tea-cup and a plate.	5.4
Magnification	The situation when the textured surface of a 3D geometric model is larger than the texture applied onto it. Either the nearest pixel can be selected, or the four nearest pixels can be interpolated.	6.1.3
Micro-texture	The fine scale detail of a texture. From a distance a surface may appear smooth, but viewed under large magnification it will have a unique texture such as scales, ripples, bumps or grooves. This is the micro-texture.	5.4

Terms	Definition	Section First introduced
Minification	The situation when the texture surface of a 3D geometric model is smaller than the texture applied onto it. Either the nearest pixel can be selected, or MIP-mapping can be used.	6.1.3
MIP-mapping	A method of improving the visual quality of a texture. MIP-mapping requires that each texture is replaced by as a set of textures, each half the size of the other.	2.1
Model space	The local coordinate system of a 3D geometric model.	5.5.2
Motion parallax	The effect of perspective that objects closer to the camera appear to move greater distance than those further away.	5.4.7
Normalmap	A method of adding detail to a texture by defining an individual surface normal for each pixel.	2.1.8
Parallax mapping	A method of implementing parallax motion for 3D surface representations.	5.4.7
Parametric coordinate	A variable used to define either a parametric curve or surface.	2.1.7
Parametric curve	A curve that is defined using a set of control points, a basis matrix and a parametric coordinate.	5.5.3
Parametric surface	A surface that is defined using a set of control points, two basis matrices and a pair of parametric coordinates.	2.2.5
Perspective projection	The method used to make a distant 3D geometric model appear smaller than the same object viewed from close by.	2.2.5
Perspective projection matrix	The matrix used to implement perspective projection	C.1
Phong lighting model	A lighting model that models specularity using a power equation based on the eye vector and the light vector.	6.1.2
Photometric image	An image used for the purposes of photometric stereo.	2.3.1
Photometric stereo	A method of acquiring the 3D surface representation of a material using a number of images.	1.2
Point set surfaces	A method of representing 3D geometry using an array of points	2.1.10
Polarogram	A method of implementing rotationally invariant texture classification	2.3.20

Terms	Definition	Section First introduced
Precision	The ability of an information retrieval System to to return only items relevant to the search query.	2.4.2
Programmable graphics accelerator board	A graphics board for a computer which can accept programs and execute them in hardware.	1.1
Projective lighting	A method of modeling the distribution of light from a light source using a texture.	5.5.2
Real-time rendering	Rendering of geometric objects and scenes at no less than 15 frames per second.	1.1
Recall	The ability of an information retrieval system to return every relevant item to the search query.	2.4.2
Relief-mapping	A rendering method which combines both normalmaps and heightmaps to texture surfaces.	1.1
Rendering method	A technique for visualizing a 3D geometric model.	1.2
Ring filter	A frequency domain filter which is only sensitive to frequency and not direction.	2.3.15
Rotation invariant	Independent of surface orientation.	1.2
Scan-line interpolation	A method of rendering objects into the framebuffer by determining the bounding leftlost and rightmost pixels of each row and processing every pxiel inbetween.	5.5.3
Scattering function	A 3D surface representation method	2.1.1
Scene-shadowing	The ability of a 3D geometric model to cast shadows onto other 3D geometric models within a 3D scene.	6.1.3
Schmid filter	A frequency domain filter that is only sensitive to selected frequencies and not direction.	2.3.17
Self-occlusion	The ability of the nearest areas 3D surface representation to appear in front of more distant areas.	2.1
Shadow mapping	A method of implementing shadows in a 3D scene by using the depth map of the scene rendered from the viewpoint of the lightsource to determine whether a point is in shadow or not.	1.2
Shadowmap	The texture used to determine whether a point in a rendered 3D scene is in shadow or not.	2.2.7

Terms	Definition	Section First introduced
Shape from shading	A method of recovering gradient and heightmap information of an object from a number of photometric images.	3.2.3
Spatial domain	The default domain for a standard texture image.	2.3
Specularity map	A texture map that allows the glossiness of a texture to specified on a per pixel basis.	4.2
Sphere mapping	A 3D surface representation that augments each pixel in the texture with the distance to the nearest points on the surface, for a number of height layers.	5.4.9
Spherical coordinates	Coordinates based on longitude and latitude.	2.1.3
Spherical harmonics	The Fourier series applied onto the surface of sphere.	2.2.8
Surface light field	A four dimensional function which defines the amount of light reflected from a uniform material.	2.1.1
Surface material	A data object which defines how a 3D geometric model should be textured.	2.1.3
Surface normal	The unit vector defined by the cross product of two partial derivative vectors (tangent vector and binormal vectors).	2.1.8
Surface orientation	The rotational position of a texture image acquired using photometric stereo.	2.3.20
Tangent space	The local coordinate system of a single point on the surface of a 3D geometric model.	2.5
Tangent space vectors	Three unit vectors defining a coordinate system for the current parametric point	5.5.2
Texton filter	A filter used to detect a fundamental unit of texture such as a point, edge or corner.	7.1
Textons	Fundamental units of texture such as a point, edge or corner.	2.3.18
Texton dictionary	A collection of textons that are capable of synthesizing a complete image	2.3.18
Texture analysis	The statistical analysis of the properties that make each texture different.	2.3.5
Texture classification	The ability for a computer system to discriminate between textures based on their statistical properties.	1.2
Texture data	The pixel information contained within a two-dimensional array of pixels.	2.3.14
Texture database	A database comprised of texture images.	1.1
Texture features	The statistical properties of a texture.	2.3.1
Texture image	A single sample of a texture.	2.3.9

Terms	Definition	Section First introduced
Texture memory	Computer memory used to store texture images.	2.1.11
Texture retrieval	The retrieval of texture samples from a texture database.	1.2
Texture space	The local coordinate system of a texture map. This is based on a number of parametric coordinates.	5.4.9
Texture-mapping	The ability to apply a texture image onto a 3D geometric model when rendered using the graphics pipeline.	1.4
Translucency	The ability of a lighting equation to represent light that has travelled through the interior of a 3D geometric model.	3.3
Transparency	The ability of a rendered fragment of texture to allow the final colour to be combined with existing colour in the framebuffer. This allows a 3D geometric model to have a transparent appearance.	2.1.11
Transparency map	A texture map that allows the level of transparency to vary per pixel.	5.4.10
Vertex shader	One of the programmable stages of the graphics pipeline. This stage transforms vertices.	5.4.4
Vertex transformation	One of the stages of a fixed functionality graphics pipeline.	5.4
View-dependent displacement mapping	A method of displacement mapping implemented using a single texture.	3.2.1
Visual quality	The photorealistic appearance of a computer generated image.	5.4.2
Wedge filter	A frequency domain filter which is sensitive to direction only.	2.3.14

Chapter 1 – Introduction

1.1 Motivation

The main motivation for the research reported by this thesis was the need of a large number of manufacturing industries to find an efficient way of implementing both online and offline virtual textile databases, with such manufacturing industries including the automobile industry, the aeronautical industry, the interactive-entertainment industry, the film industry, and in particular the textile manufacturing industry. Traditionally, a textile manufacturing company presents the range of textile products available through the distribution of textile sample books. These books can range from small paperback albums of swatches of fabric to four-inch thick A3 sized binders containing hundreds of fabric samples. The distribution of such textile sample books also poses a problem. For a single manufacturer, supplying every customer with a new set of textile sample books will also incur the cost of national and international transportation and delivery. Due to the time required to collect, bind and document all textiles available, publication of such documents is restricted to an annual or seasonal release date. In addition, customers are only allowed to borrow these books for a short period of time, rather than allowed to keep them permanently. While alternative methods such as paper catalogues are readily available, they still have the disadvantage of requiring the physical distribution of such catalogues to all interested customers.

Fortunately, the rapid availability of Internet based applications such as standalone and embedded web browsers provides an alternative solution. Rather than distribute the textile catalogues physically, it is now possible to place an entire textile catalogue online, and allow potential customers to view or download the catalogue without the associated physical distribution and collection costs. The use of such catalogues also has the advantage that search engine technology can be used to enable customers to search for particular textile patterns, thus effectively creating a *texture database*.

However, there are two problems with such online catalogues. The first problem is that the search features of such databases are still very basic, with the simplest feature consisting of a simple keyword search for a colour or associated name, and the most complex search feature being able to search for the closest colour. The second problem is that neither of the existing methods of presenting textile databases (physical books or web-based online catalogues) solves the problem of allowing the user to visualize individual textile samples under changing lighting conditions. Such examples of changing light conditions include the customer physically changing the orientation of the textile sample, or moving the textile sample past a light source (sunlight, light bulb, or fluorescent tube). Some textile manufacturers have attempted to provide 3D views of their products by photographing their products from multiple directions and allowing the user to switch between these images, thus creating the illusion of rotation. While making the user-interface interactive, it does not solve the problem of allowing the user to change or control the lighting conditions of the desired textile sample.

A solution to this problem comes in the form of the combination of *shape-from-shading* algorithms [Woodham1980], *integration methods* [Frankot1988] and the availability of consumer level *programmable graphics accelerator boards*. With the use of *shape-from-shading* algorithms, it has become possible to acquire the *3D surface representation* of a textile (an *albedo* image, two *gradient fields* and/or a *height field*). This *3D surface representation* can then be used in conjunction with a variety of rendering techniques such as Blinn bump-mapping [Blinn1978] [Cook1982], *relief-mapping* [Olivieria2000], per-pixel *displacement mapping* with distance functions [Hart1996] and shell mapping [Porumbescu2005] to enable the user to interactively view textured *3D geometric models* in *real-time*. With, the availability of the additional data provided by *3D surface representations*, new methods of searching *texture databases* are now required, in particular, methods that are capable of finding similar samples regardless of orientation (*rotation invariant*) are required.

Thus, there are three objectives for this thesis. The first objective is to determine the most suitable *3D surface representation* for the acquisition, retrieval and visualisation of textiles. The second objective is to determine the most effective way of searching a textile database composed from a set of *3D surface representations*, and the third objective is to determine the most effective way of presenting the textile *image data* to

the user through the use of *real-time rendering* techniques provided by *programmable graphics accelerator boards* and through the use of the appropriate selection of *geometric objects*.

1.2 Scope of the research

The research carried out by thesis includes the following:

- Identifying the most suitable method of representing the *micro-geometry* of textile samples acquired using *photometric stereo* techniques (the *3D surface representation*).
- Identifying the most suitable *rendering method* to present the acquired *3D surface representations* of textile samples (the *micro-geometry*) combined with the 3D geometry of an object (the *macro-geometry*) to the user as realistically as possible in *real-time*, using current generation *programmable graphics accelerator boards*.
- Identifying the most suitable methods of implementing the feature extraction and similiarity matching modules of a modern information retrieval system based upon *rotation invariant texture features* of *3D surface representations.*

The first area of investigation involves identifying the most suitable method of representing the unique texture characteristics of *3D surface representations* of textile samples that is both compact and flexible enough to be used for similarity matching using *rotation-invariant texture retrieval* methods and *real-time 3D visualisation* using current graphics hardware. The second area of investigation involves identifying the most suitable way to present these *3D surface representations* to the user in terms of memory usage, *real-time* performance, *lighting models* and *shadow-mapping* techniques and *3D geometric objects* running on the current generation of *programmable graphics accelerator boards*. The third area of investigation is the identification of the most suitable method of *rotation invariant texture classification* in order to implement a *texture retrieval* system that will allow the user to select textures based upon similarity. Because the three objectives of this thesis are based upon the acquisition, visualisation and retrieval of textile samples, implementing a *real-time* physics system modeling the animation, draping, folding and wrinkling of cloth is beyond the scope of this thesis, and we do not perform a survey in this area.

Note that in this thesis we focus our attention on the evaluation of the capabilities of *texture features* for *texture retrieval* rather than examining the issues concerning the design and implementation of a complete *texture retrieval* system such as that provided by the Ferret framework [Lv2006].

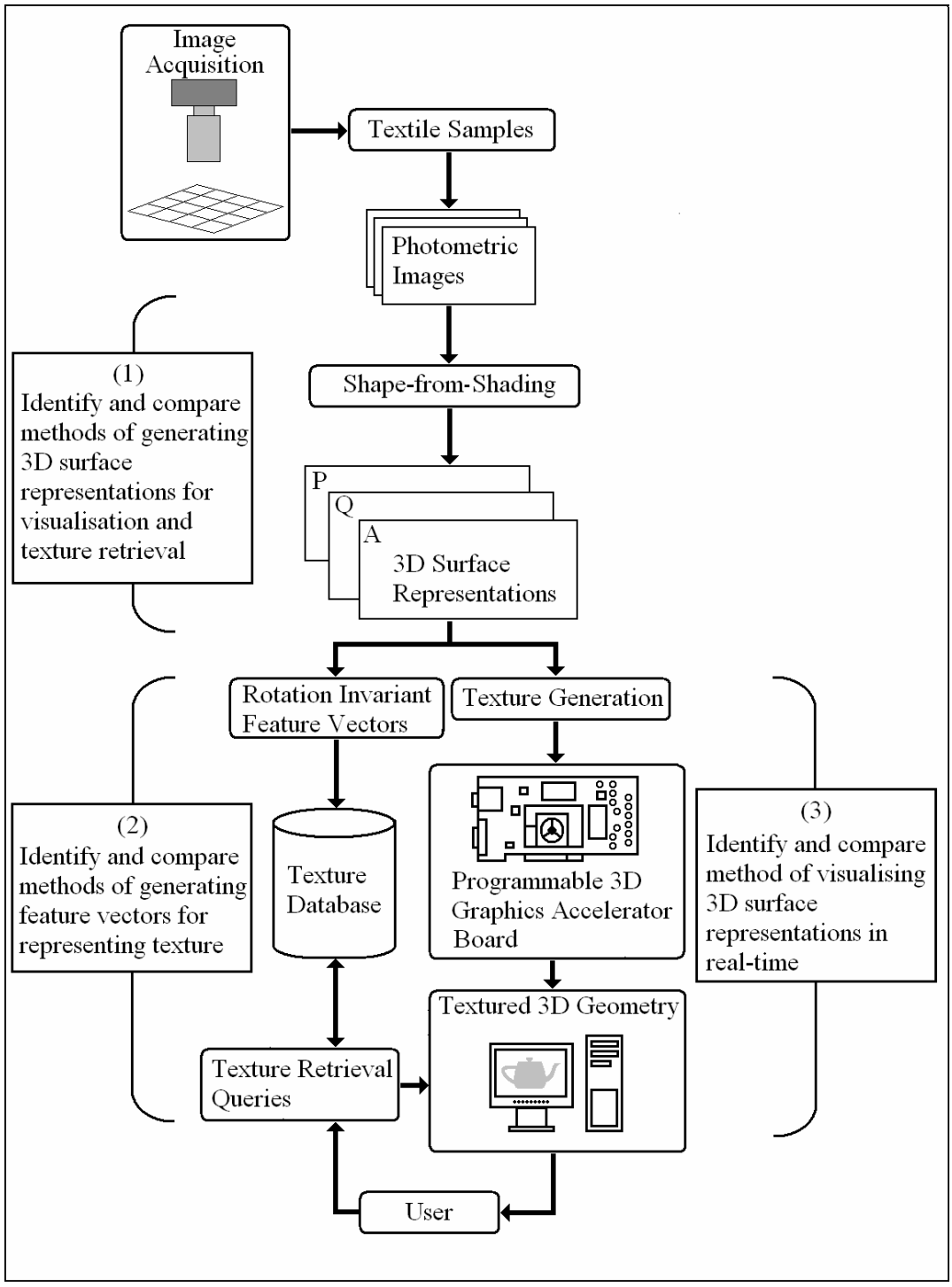


Figure 1: The scope of research conducted by this thesis

1.3 Thesis Organisation

We have organized this thesis is into seven chapters (Figure 2). In Chapter 2, we provide a review of the literature considered most relevant to our research into acquiring *3D surface representations*, presenting these *3D surface representations* to users via *real-time rendering methods*, and retrieving such *3D surface representations* using *rotation-invariant texture retrieval* methods. In Chapter 3, based upon our review, we describe a representation suitable for use with both the acquisition of *3D surface representations* of textiles using *photometric stereo* techniques. In Chapter 4, we conduct a brief review of the use of *photometric stereo* to acquire *3D surface representations* which are suitable for use with *texture retrieval* and visualisation of textiles. In Chapter 5, we identify ten candidate methods of visualizing these *3D surface representations* in real time using *programmable graphics accelerator boards* and identify the one most suited to our needs. In Chapter 6, we describe how we present the textile samples to users in *real-time*. In Chapter 7, we identify and evaluate ten different methods of implementing *texture retrieval* with *3D surface representations*. In Chapter 8, we summarize the research carried out by this thesis and describe the applications of our research in commercial visualisation applications.

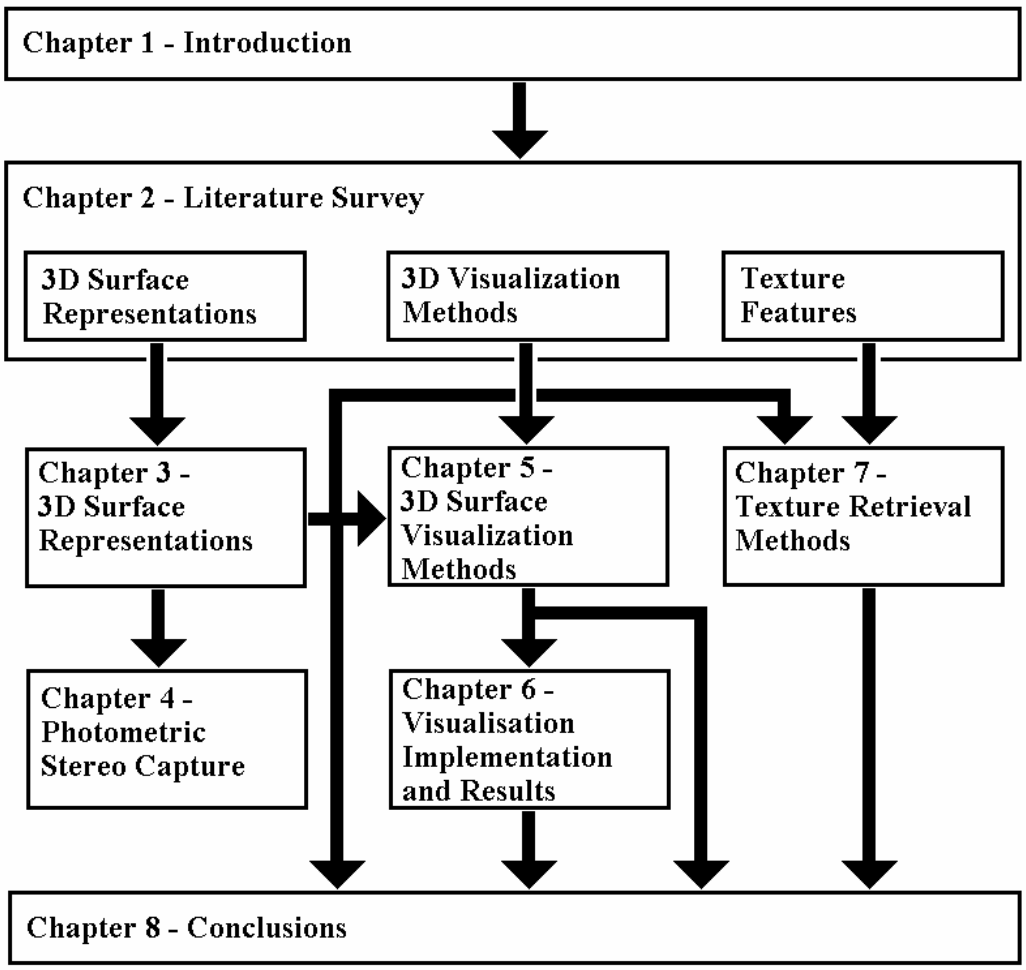


Figure 2: Logical structure of the thesis

1.4 Original Work

We believe that this thesis contains the following original work:

1. In Chapter 6, we describe a method of visualizing the *3D surface representations* that combine together procedural surface generation of *parametric surface*s to represent the *macro-structure*, *texture-mapping* and *relief-mapping* to represent the *micro-structure*, dynamic point and infinite light sources and hardware *shadow-mapping*, all integrated together in order to achieve interactive 3D *real-time rendering* of textile samples. To our knowledge, this is the first time that all such methods have been combined together in a single application.
2. We also believe that this thesis, combined with our previous publications, makes an important contribution to the research fields of *texture retrieval* through the evaluation of a set of *texture features* that exploit both colour and surface relief periodicity information and that use a common data set with that used by (1) above.

Chapter 2 - Literature Survey

In this chapter, we perform a survey of the research literature relevant to the areas of investigation identified and described in Chapter 1. These are:

- Identification of candidate *3D surface representations* most suitable for use with fast, time-efficient acquisition methods, *rotation-invariant texture retrieval* and *real-time 3D visualisation* (section 2.1).
- Identification of the most suitable way to implement the *3D visualisation* of these *3D surface representations* of textile samples using *real-time* lighting and shadow generation techniques with *3D geometric models* that run on *programmable graphics accelerator boards* (section 2.2).
- Identification of the most suitable methods of implementing the *texture retrieval* of textile samples using *rotation- invariant features* (section 2.3).

As the objective of this thesis is to implement a textile database system that integrates 3D *real-time* visualisation with *rotation-invariant texture retrieval*, it is essential that each *texture database* entry should be in a format that can be used to both generate *rotation-invariant feature vectors* and generate textures *real-time 3D visualisation*. This is the purpose of the *3D surface representation*. Because we wish to present the *micro-geometry* of textile samples as photorealistically to users as possible, we choose to investigate suitable methods of visualising *3D geometric objects* with shadows. As we also wish users to be able to select textiles based on similarity to each other, we choose to investigate methods of generating *feature vectors* from the *micro-geometry* and the colour information. We begin our survey by reviewing candidate *3D surface representations*.

2.1 Review of Candidate 3D Surface Representations

As we mentioned previously in Chapter 1, in order to successfully implement the *real-time 3D visualisation* and *rotation-invariant texture retrieval* stages of this thesis, it is necessary to have a common data format that is suitable for use by both of these stages. We refer to this data format as the *3D surface representation*.

For the purpose of representing textile samples, we believe that it is important that our representation can encode the *micro-geometry* and colour of the textile samples and that this should be combined with *macro-geometry* information on the overall shape of the sample as double curvature and other undulating shapes can be effective for presenting the textile properties to the user.

To satisfy these requirements, we perform a literature survey of all such documented representations, and choose the data format that is most suitable to the following criteria, arranged in the order of importance:

- Must be able to be visualised in *real-time*
- Must be able to be acquired using time efficient acquisition methods
- Must be able to represent the textile at a suitable high resolution
- Must be compact in the use of system memory or memory usage
- Must be able to represent both the colour and *micro-geometry* of textiles samples ie. represent the variation in appearance all across an image
- Must be suitable for the integration with a 3D surface representation that will be used to define the overall shape of the textile sample
- Must be able to visualize the *self-occlusion* and *self-shadowing* of the *micro-geometry* of each textile

As the goal of this thesis is to have an interactive database, we require that the system is able to operate in *real-time*. As we require our samples to be as photorealistic as practical, it is essential that the textile samples can be acquired using through some economic acquisition process and that the acquisition process is also able to present the *micro-geometry* of the textile samples at a suitable high resolution to the user. At present, the two most common screen resolutions are 1024x768 pixels and 1280x1024,

while high end monitors may have resolutions as high as 1920x1200. Since we wish to present the *micro-geometry* to the user, this requires a resolution matched to that of the screen. Since the *texture mapping* functionality of *programmable graphics accelerator boards* operates on dimensions based upon the power of two in order to implement *MIP-mapping*, this imposes a constraint on the size of textile sample. For this reason, we choose 512x512 as the default resolution for all textile samples in this thesis.

At present, there are a large variety of ways of acquiring both the *albedo* and *micro-geometry* of surfaces depending on the dimensions of the target sample. For geographic terrain hundreds of kilometers in size, the combined use of satellite or airborne photography, radar and laser ranging are used to acquire this data. For 3D objects such as buildings and artifacts, laser scanning can be used. Alternatively, silhouette scanning, stereoscopic and photometric methods are also available. Silhouette scanning builds up a visual hull of an object by analyzing the silhouette of the object from multiple camera angles. Stereoscopic methods make use of multiple cameras while photometric methods use a single camera, but make use of multiple light sources.

We define the *albedo* of a textile sample as the colour information of a textile under ambient light conditions where the intensity of light is identical in every direction. We define the *micro-geometry* of a textile sample as the variations in height on the surface of the material that cause the effects of *self-shadowing*, *self-occlusion* and *rough-edge silhouettes* on the appearance of any *geometric object* that the *textile sample* is applied to. As one of the objectives of this thesis is to create a virtual textile database with textile samples that are as photorealistic as possible to the original textile, it is essential that both the *albedo* and the *micro-geometry* are represented for visualisation purposes. The *micro-geometry* must also be represented in order to allow *rotation-invariant feature vectors* to be generated for *texture retrieval*.

2.1.1 Selection of the 3D Surface Representation

During the past century, researchers have developed many different *reflectance functions* which aim to capture particular attributes of the materials under study. These attributes can include the time between absorption and emission of individual photons,

the way in which light received from a particular direction is reemitted, with the distribution being measured across the surface of a hemisphere, the variations in appearance across the surface of a material or *micro-geometry*.

This is in contrast to the *macro-geometry* of an object which consists of the physical 3D geometry of the object itself and is defined by representations such as explicit and implicit surfaces, triangulated meshes, and more recently, *parametric surfaces* such as Bézier patches and trimmed NURBS surfaces [Bézier1974] [Bézier1983] [Piegl1997]. Initially designed for use in the automobile industry, *parametric surfaces* have rapidly found applications in the aerospace, animation and textile industries. While Bézier surfaces are limited to representing solid rectangular or triangular patches, NURBS surfaces can represent 3D geometry consists of curved shapes with trimmed edges and holes, but at the cost of requiring considerable more processor time. Because the mathematics for evaluating the 3D geometry on Bézier surfaces is well documented and straightforward [Bézier1974] [Bézier1983] [Piegl1997], we will not discuss them further in this chapter except to note that the geometry is an important issue which we address in Chapter 5. Therefore, we focus our attention here on the representation of the *micro-geometry*.

As the data set for each *reflectance function* is measured discretely, each additional variable will increase the size of the data set accordingly. Thus, there is a trade-off between the complexity and accuracy of the *reflectance function* against the size of the data set. Because of this, many researchers have simplified more complex *reflectance functions* by only taking into account angular distance between the direction of received or emitted light (isotropic), or by assuming that the material has no *micro-geometry*. Thus, the goal of this review is to identify the *reflectance function* that offer the most accurate representation with an economical use of memory that can be adapted for use as a *3D surface representation*. We begin this review by presenting a version of the "Hierarchy of reflectance functions" originally presented by Müller [Müller2004b] and extended to include both *general functions* and *scattering functions* (Figure 3).

In this diagram, each *3D surface representation method* is placed in the graph according to the number of parameters required to define that function. At the top of the chart is the General Function which takes into account both the time delay and the

change in wavelength between the absorbed and emitted photons as well as the absorption and emission directions. Below are the *Scattering Function* and the BSSRDF, both of which assume the time between absorption and emission is instantaneous and that there is no change in photon absorption and emission wavelength. Each of these systems requires a ray-tracing system to implement using either using one or more CPU's, or a *programmable graphics accelerator board*.

Below this level are the BTF and BSSRDF, each of which makes different trade-offs. The BTF assumes that the photon wavelength remains constant between emission and absorption but that the appearance will vary across the surface, while the BSSRDF assumes that the appearance will remain constant across the surface, but that the appearance will vary depending upon emission and absorption directions. Below these are the Surface Reflectance Field and Polynomial Texture Map, which assume that the appearance will vary across the surface, but only take into account the direction of the emitted photon.The *Surface Light Field*, Homogeneous BRDF, BTDF and Diffuse Subsurface Reflectance Functions, all assume that there are no variations across the surface of the material, but that only either the direction of the absorbed or emitted photon are known. A further simplification of the Homogeneous BRDF is the isotropic BRDF which only takes into account the angle between the direction of the absorbed and emitted photon. All of these methods can operate on current *programmable graphics accelerator boards* using a suitable data compression algorithm, but have the disadvantage of being extremely expensive and time consuming to acquire, often requiring several hundred camera units to acquire each textile image simultaneously.

At the very bottom of the diagram are the simplest models of all. The *relief-map* and *bump-map* assumes that only the intensity of light will vary due to variations in surface *micro-geometry* through Lambertian or specular reflection, while the texture map assumes that the emission wavelength remains constant regardless of direction of emission. Below that is the gloss-map which assumes that the material is a uniform colour with no variations across the surface. Each of these systems can run on entry level graphics boards. We now proceed to describe and evaluate each of these in detail below:

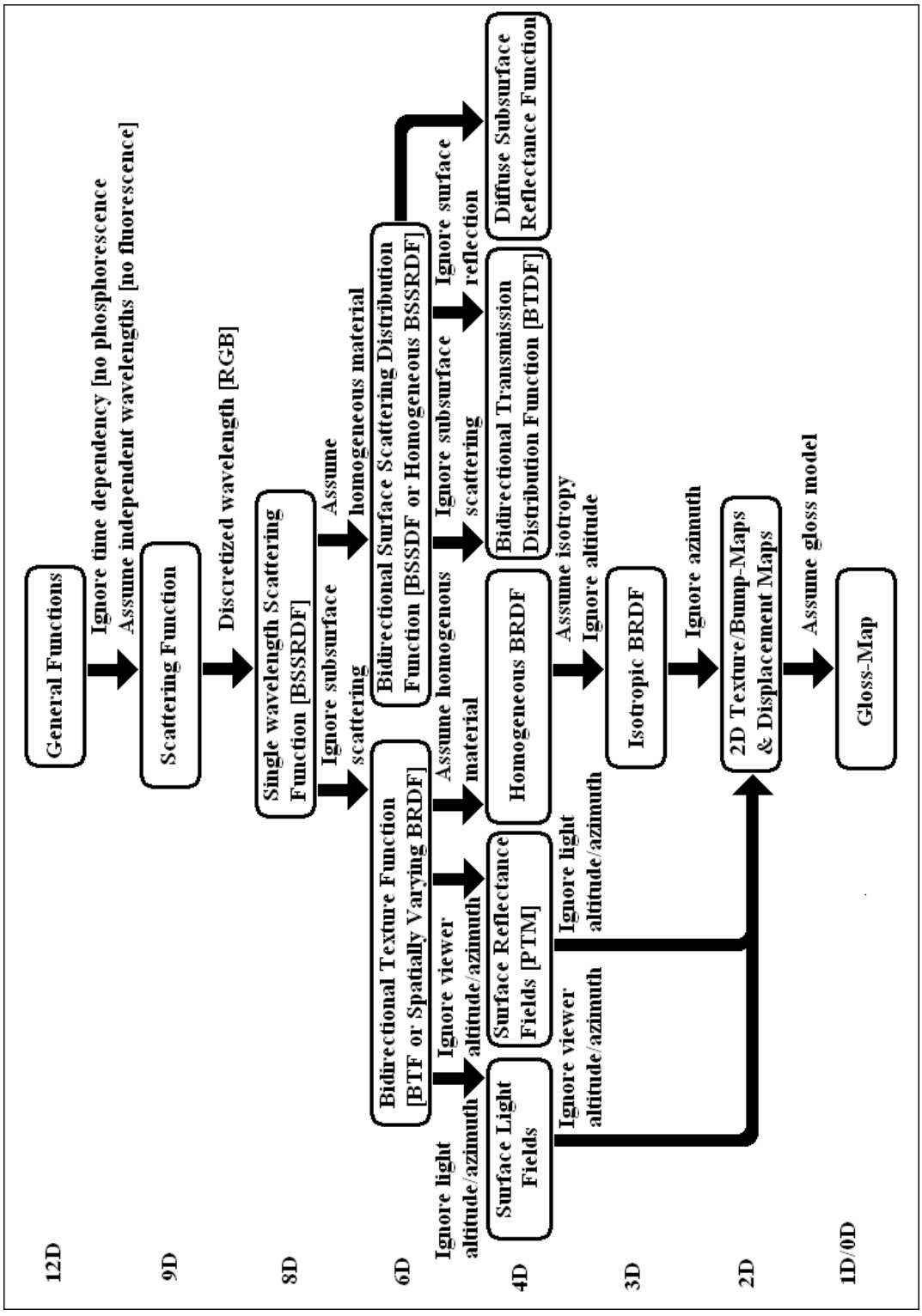


Figure 3: Hierarchy of 3D surface representation methods

(based upon the paper by Müller [Müller2004b])

2.1.2 General and Scattering Functions

The *general lighting function* [Müller2004b] attempts to model the complete behaviour of light at the molecular level taking into account wavelength, direction, location and time of both the photon being absorbed and the photon emitted. This is of particular use for modeling the behavior of luminous materials which continue to emit light even after the original illumination source has been removed. The *scattering function* simplifies this function by assuming that the times of absorption and emission are identical and that the photon wavelength does not change. The advantages of these two methods are that they are both extremely accurate in the modeling of the exchange of photons. However, the first disadvantage with both of these methods is that the amount of time required to perform these calculations through ray-tracing is extremely prohibitive and not suitable for *real-time rendering*. The second disadvantage is that it is not possible to economically acquire this behaviour without complex measuring equipment. Because of the use of ray-tracing, this method is not suitable to the needs of this thesis.

2.1.3 Bi-directional Scattering Surface Reflectance Distribution Function (BSSRDF)

The BSSRDF attempts to reduce the complexity of the *General and Scattering functions* by decomposing the wavelength of each photon into three colour bands. A further reduction can be achieved by making the assumption that the *surface material* is homogeneous. By making these assumptions, performance gains in the computer simulation of such materials can be gained. A further simplified version of the BSSRDF is the Bidirectional Scattering Distribution Function (BSDF) which is a four parameter function. The two inputs are the *spherical coordinates* of the direction of incident light, and the outgoing reflected or transmitted light. The output of the function is the ratio of the light energy of the two inputs. The BSDF unifies two components.

- The BTDF (Bidirectional Transmission Distribution Function)
- The BRDF (Bidirectional Reflectance Distribution Function)

The BTDF is used to model light refracted through the surface of the material, while the BRDF is used to model light reflected from the surface of the material. As identified by Jensen [Jensen2001], the advantage of the BSSRDF is that it provides an accurate

model of light reflection, while the disadvantage of the BSSRDF are that this method still requires a ray-tracing system for practical usage. Another disadvantage of the BSSRDF is that measurement of the BSSRDF requires a calibrated measurement system that not only requires the measurement of the resulting light intensity, but also the directions of both the light source and receiver. The use of ray-tracing and the calibrated measurement system thus makes the BSSRDF unsuitable for our needs.

2.1.4 Bi-directional Reflection Distribution Function (BRDF)

The BRDF takes two parameters as input; the first is the direction of the viewer in *spherical coordinates* and the second is the direction of the light source also in *spherical coordinates* [Nicodemus1977]. Thus accurate representation of the BRDF requires a four-dimensional table of data consisting of measurements of every combination of angle of emitted and received light. Due to the size of this table, substantial research has been performed in the field of data compression of the BRDF. However, as a survey of BRDF compression methods is beyond the scope of this thesis, we provide a chronological index in Appendix D.1. The advantages of using the BRDF include a more accurate *lighting model* including the ability to implement *anisotropic reflections* and that it can be used with current *programmable graphics accelerator boards*. However, there are also several disadvantages with using the BRDF. The first is that the BRDF does not capture the fine scale variations in texture that occur in natural *surface materials.* To acquire this information, we would require the spatially varying BRDF (see 2.1.5). Thus it is not possible to model *self-shadowing*, *self-occlusion* or *rough-edge silhouettes*. The other disadvantage of the BRDF is that acquisition of the data requires a custom measurement system that is capable of moving both the light source and photo-detector. Since the goals of this thesis are to visualize the *micro-geometry* of textile samples in real time, we conclude that this method is not suitable to our objectives.

2.1.5 Bi-directional Texture Function (BTF or Spatially Varying BRDF)

Dana introduced the BTF (Bi-directional Texture Function) as a means of extending the BRDF to capture the *fine scale surface detail* with varied illumination [Dana1999]. The

advantages of this method are that the *micro-geometry* of the *surface material* under varying lighting conditions can be reproduced.

Unfortunately, there are two disadvantages to this method. The first disadvantage is that the method of measurement of the BTF described in this paper is slow and requires specialist equipment which includes a personal computer with a RGB framegrabber, a robot arm with a photometer and a halogen bulb with a Fresnel lens (Figure 4).

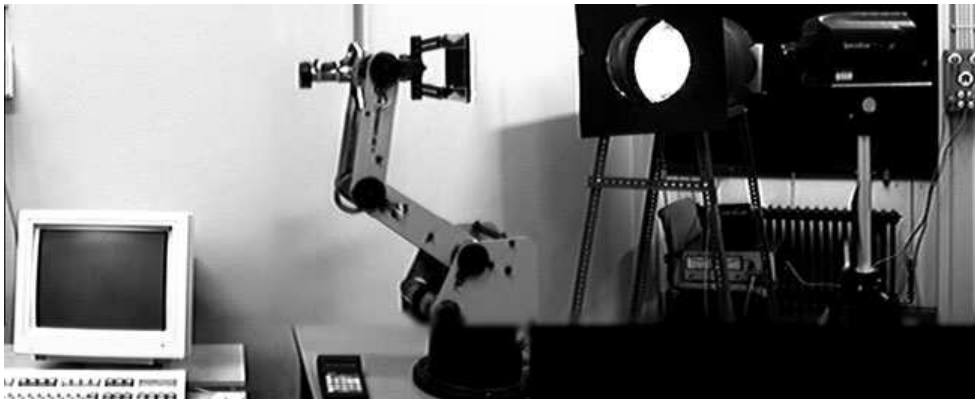


Figure 4: Measurement setup for the BTF with textile sample on robot arm

(from [Dana1999])

The second disadvantage of this method are that large amounts of memory are required to store each sample when no data compression is used, and that the rendering speed of the system is reduced to below *real-time* frame rates if data compression is used. Stored in raw data form, each BTF data set consists of 6561 *photometric images* (81 camera positions with 81 light source positions) at 800x800 pixel resolution in RGB data form. This results in a raw data memory requirement of between 733 Megabytes and 5.3 Terabytes of data [Filip2005]. However, there is a trade-off between the level of compression and the resulting rendering frame rate, with higher levels of data compression requiring more time to render. Even with the best data compression algorithms, storage of a single BTF sample will still require between 4 and 30 Megabytes of data, and also require several hours of processing time to compress the data. Due to the difficulty of this problem, resesearch into such compression algorithms for multi-dimensional data sets is ongoing [Filip2004] [Filip2005] [Filip2008]. For this reason, we consider this method unsuitable for the objectives of this thesis.

2.1.6 Surface Light Fields and Surface Reflectance Fields

Two alternative ways of reducing the amount of memory required to store the BTF involve reducing the BTF from six to four dimensions by ignoring the *spherical coordinates* of either the light source or the viewer. Ignoring the *spherical coordinates* of the light source produces the *surface light field* [Gershun1936] [Wood2000]. Ignoring the *spherical coordinates* of the viewer produces the *surface reflectance field* [Weyrich2005]. Levoy described a method of acquiring the *surface light field* using a gantry system comprised of a rotation hub of light sources, a rotating platform and video camera [Levoy1996]. We present a diagram of the setup in (Figure 5).

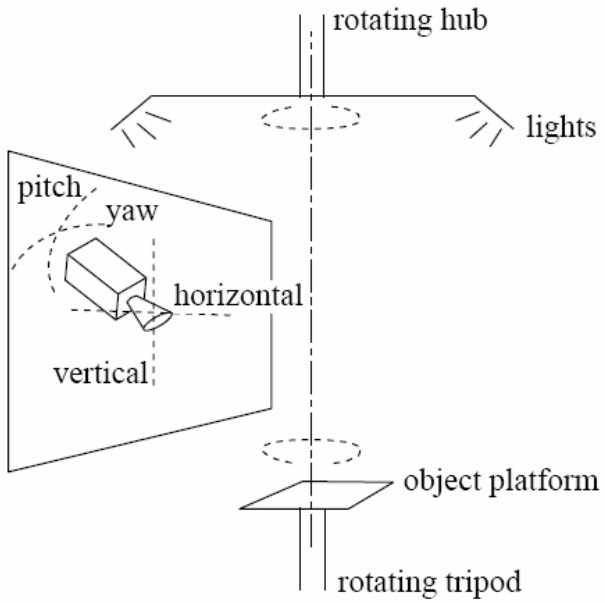


Figure 5: Capture equipment for surface light and reflectance fields

(from [Levoy1996])

Ignoring both the *spherical coordinates* of the viewer and light source produces standard *texture-mapping*. The advantages of these methods are the simplification of the *lighting equation* which in turn improves performance and pre-calculation. However, these methods have the disadvantage of no longer having an accurate *lighting model* which takes into account both the relative orientation of the local light source and the light reflected from the *micro-geometry* of the surface. Because of the necessity for an

an automated system and associated specialized setup to acquire photometric data, we do not consider this method suitable for this objectives of this thesis.

2.1.7 Polynomial Texture Map (PTM)

Malzbender proposed the PTM (Polynomial Texture Maps) as a solution to reduce the memory requirements of the BTF [Malzbender2001] and *surface reflectance field*. This had the benefit of allowing textures to be rendered at *real-time* frame rates. Using this method, a large set of images of the surface are acquired using a combination light sources or cameras. The PTM is generated through the construction of a bi-quadratic polynomial for each pixel.Within the texture, ther bi-quadratic polynomial is stored within each pixel using either LRGB (Luminance, Red, Green and Blue) or RGB formats. With the LRGB format, nine bytes per pixel are required, while the RGB format requires eighteen bytes per pixel (six polynomial coefficients for each colour channel). Acquisition of the PTM is performed through either manual placement of light sources and a stationary digital camera (Figure 6) or an automated camera system which can be purchased directly from Hewlett Packard (Figure 7). The advantages of this method are the reduced memory requirements and the ability to render the texture from any lighting angle by setting the appropriate *parametric coordinates* to the PTM. There are several disadvantages to this method. The first is that this method does not implement *rough-edge silhouette* generation or *self-occlusion*, while the second is that this method requires custom photographic equipment to acquire a large set of photographs. As this does conform to our criteria for economic photographics methods, we do not consider this method suitable for the goals of this thesis.

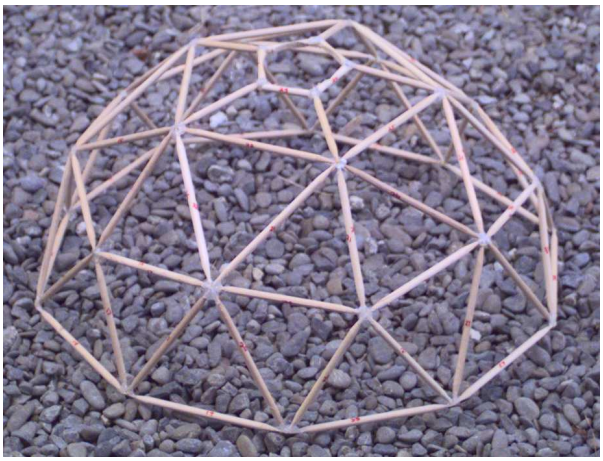


Figure 6: Template system for manual placement of light sources over sample

Figure 7: Automatic camera system for capturing PTM's

2.1.8 Texture-mapping and Blinn Bump mapping

Blinn proposed a solution to the problem of adding *fine scale surface detail* without changing the underlying geometry by introducing the concept of *bump-mapping* [Blinn1978]. With this method, the surface of an object has *fine scale surface detail* added by perturbing the *surface normal* of each visible point on the surface before the lighting calculation. This method has the advantage of only requiring a mathematical function to define the perturbation of the *surface normal* and allows additional detail to be added to a scene without affecting rendering speeds. We present images from Blinn's paper below (Figure 8) (Figure 9). Cook proposed an even more economic solution in that instead of perturbing the *surface normal,* the original *surface normal* is replaced by a lookup call into a texture map representing encoded *surface normals* [Cook1982].

Cook refers to this as a *normalmap*. The advantages of implementing both these methods are that only a single mathematical equation is required to modify or replace the *surface normal*. The disadvantages with both of these methods are the inability to implement *self-shadowing*, *self-occlusion* or *rough-edge silhouette* generation. The lack of *self-shadowing* means that raised areas of the surface will not create shadows on those areas hidden from the line-of-sight of the light source, and the lack of *self-occlusion* means that raised areas will not obscure those areas hidden from the line-of-sight of the observer. The lack *of rough-edge silhouette* generation means that while the surface may appear visually bumpy towards the observer, the surface will still maintain the appearance of underlying polygon geometry at the boundaries of the object and the background.

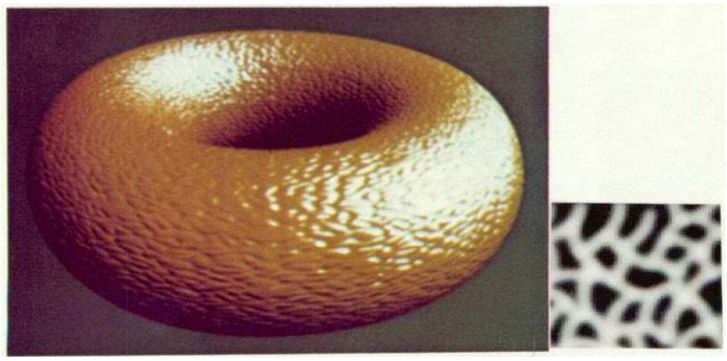


Figure 8: Image rendered using bumpmapping along with associated bumpmap

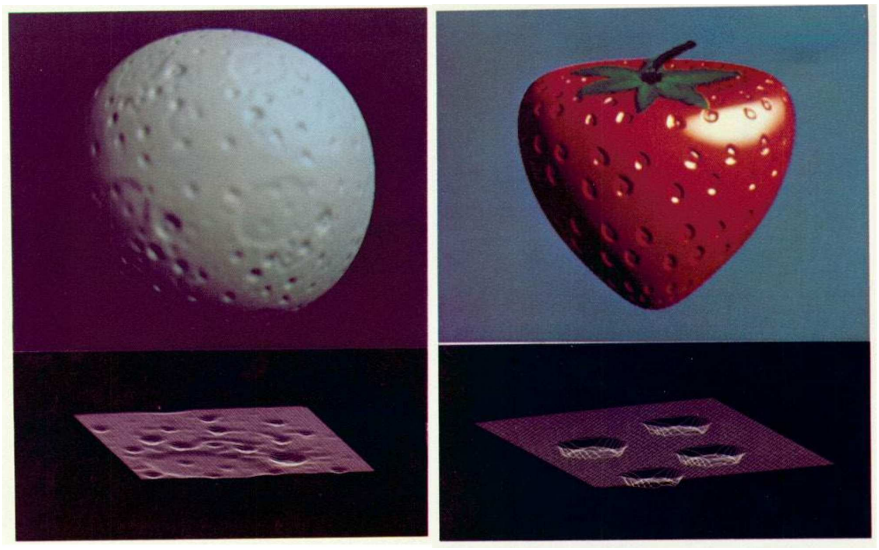


Figure 9: Images rendered using bumpmapping along with associated bumpmap

A further extension to the *texture-mapping* combined with bump-mapping is the use of *gloss maps* [Blythe1999]. With this method, the *alpha channel* of each pixel in the base texture (or *albedo*) is used to store a coefficient which represents the amount of specular reflection of that texture region. This value can range from 0.0 for no reflection, to 1.0 for full reflection. One practical use for the use of this method when applied to the visualisation of textiles is to be able to represent the different specular lighting properties of individual fabric fibres. The advantage of this method is that all the information required to store the appearance of the material can be stored compactly in a pair of *texture images*, the first image stores the *albedo*, while the second image stores the orientation of the individual material samples. The disadvantage of this method is that it is not as accurate as other more advanced *lighting models* such as the BSSRDF, BTF, or BRDF, and in particular does not encode relief data.

2.1.9 Relief-mapping

A further improvement upon the use of Blinn *bump-mapping* is the use of *relief mapping* [Oliveria2000b]. This method improves upon Blinn *bump-mapping* by using per pixel relief data in addition to that required for *bump-mapping* and taking advantage of *programmable graphics accelerator boards* to perform ray-casting on a per-pixel basis, with the desired effect of adding detail to a scene without requiring additional geometry (Figure 10). Policarpo extended this method to run on *programmable graphics accelerator boards* [Policarpo2005]. This method has several advantages. The first advantage is that it presents the *3D micro-geometry* to the user along with *self-occlusion* and *self-shadowing*. The second advantage is that it makes used of *MIP-mapping* to avoid aliasing effects. Another advantage is that this method can present *rough-edge silhouettes*, but only by discarding those pixels which lie within the silhouette edge of the geometric object.

Figure 10: Conventional texture-mapping (left) and relief mapping

(from [Oliveria2000b])

Figure 11: Teapot rendered using relief-mapping

(from [Policarpo2005])

2.1.10 Point-set surfaces (PSS)

Levoy and Whitted introduced the concept of *point set surfaces* (PSS) for the representation and visualisation of complex three-dimensional geometric objects [Levoy1985] and later adapted these algorithms to run on high-end graphics hardware [Rusinkiewicz2000]. Rather than representing a three-dimensional geometric object as a triangulated or polygonal mesh combined with one or more texture maps, point set surface *rendering method*s represent the geometry solely as a list of *geometric points*;

the *point set surface* (the PSS). Each *geometric point* within this set consists of a three dimensional vertex coordinate, colour, specular lighting and tangent-space information. *Point set surfaces* can be acquired either from laser scanning of a real world object, or from a pre-existing polygon mesh object by rendering the object using an orthographic projection and then recovering sampled points from each pixel of the framebuffer. Each individual point can be rendered either as a single pixel, a square block of pixels scaled by distance, a Gaussian smoothed circle scaled by distance, or an ellipsoid scaled and rotated according to the alignment of the outward normal of the point. We present images rendered using point set samples in (Figure 12) and (Figure 13).

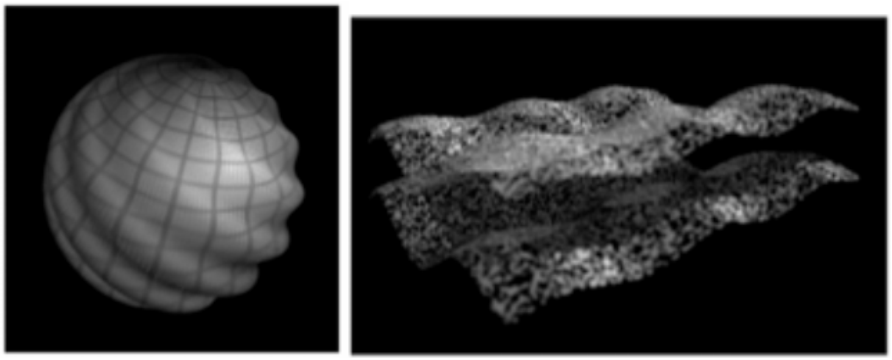


Figure 12: Images rendered using point set surfaces

(from [Levoy1985])

For pure software rendering, the advantage of the point set surface is that many of the complex stages of the traditional graphics pipeline are eliminated. These include the polygon clipping, triangle rasterisation, *texture-mapping* and bump-mapping stages [Grossman1998]. The disadvantages of "splatting" techniques are that "splatting results in poor image quality under magnification" and that "splatting-based rendering algorithms typically do not account for secondary effects such as shadows or reflections" [Adams2005]. The other disadvantage of the "splatting algorithm" is the slow rendering speed. Even with the latest *programmable graphics accelerator boards*, recent research only achieves a frame rate of 28 frames/second (Stanford Bunny dataset) and 20 frames/second (Horse dataset) with no shadowing effects, and 10 frames/second (Stanford Bunny dataset) and 11 frames/second (Horse dataset) when shadowing is implemented [Tejeda2006] [Tejeda2007]. As the goal of this thesis is to render geometry with shadows in *real-time*, we do not consider this method suitable to our needs.

Figure 13: Statue of an angel represented as a point set surface

(from [Alexa2001])

2.1.11 Summary of micro-geometry and colour representation

We have reviewed the literature on *lighting models* and equations and have made the following observations with each method:

General and Scattering Functions provide the most accurate modeling of the absorption and emission of light at different time intervals but at the cost of requiring a ray-tracing system to model every single photon wavelength [Müller2004b]. Thus, these methods do not operate in *real-time* and so are not suitable for our needs.

The BSSRDF attempts to reduce the complexity of the *General lighting function* and the *Scattering function* by using just three photon wavelengths to model the RGB nature of computer displays [Jensen2001]. However, this still requires a ray-tracing system. Thus, this method is not suitable for our needs.

The BTF/SLF/SRF methods represent each sample through a set of images which are combined together mathematically [Weyrich2005]. While this method can operate in *real-time*, between 700 Megabytes and 5 Gigabytes of memory are required to represent a single sample in uncompressed data form and between 4 and 30 Megabytes in compressed data form, along with several hours of processing time to compress the data. The data capture process is very time consuming and requires specialized equipment which typically includes one or two hundred cameras). Because of the data requirement and the time consuming capture process, we do not consider this to be a suitable for our needs.

Point-Set Surfaces represents the *3D surface representation* simply as an extremely large number of sample points, in the range of millions of billions. While this method is able to represent the *micro-geometry* of a sample, it does not operate in real time and thus would be difficult to use for *texture retrieval* purposes. Thus we do not consider this method suitable for our needs.

The PTM represents each pixel as a polynomial equation derived from a large set of *photometric images*, but at the cost of requiring specialized equipment that is not

available off-the-shelf [Malzbender2001]. Thus, this method is not suitable for our needs.

Texture mapping requires only a single *albedo* image to represent each textile sample, which is the most economical method of representing data, but at the cost of not being able to present the *micro-geometry* of the textile sample to the user [Catmull1974]. Lighting calculations are based upon the interpolation of vertex normals, rather than on a per-pixel basis. However, as this method benefits from hardware acceleration and operates in *real-time*, we choose to investigate this method further.

Bump-mapping, in contrast to *texture-mapping*, requires both an *albedo* image and a *normalmap* image in order to represent a textile sample and implement per-pixel lighting calculations [Blinn1978]. This has the benefit of being able to operate in *real-time*, but at the cost of requiring an entry-level or mid-range *programmable graphics accelerator board* and not being able to perform *self-occlusion* or *self-shadowing*, or rough edge silhouette generation. For this reason, we consider the *bump-mapping* method suitable for further investigation.

As an alternative to the previous two methods, *Relief mapping* requires both an *albedo* image and a combined *heightmap* and *normalmap* in order to store the *3D surface representation*, but at the cost of requiring a high-performance *programmable graphics accelerator board* [Policarpo2005]. However, as this method is able to perform *self-occlusion* and *self-shadowing* and does provide a rough-edge silhouette, we consider it suitable for further investigation.

Thus, we consider *texture-mapping*, *bump-mapping* and *relief-mapping* to be the three methods suitable for further investigation. Each of these methods has the ability to operate in *real-time*, have economic memory usage and the data is economically available via *photometric stereo*. However, only *relief-mapping* is able to present the *micro-geometry* of a textile sample to the user. Thus, there is a trade-off between requiring a basic 3D graphics card, and using a high-performance *programmable graphics accelerator board* in order to present the *micro-geometry* to the user and thus implement *self-shadowing* and *self-occlusion*.

We summarize all of the above in the following table, where the terms are now explained as follows. The term *real-time* refers to the ability of the technique to render a complex *3D geometric model* at a responsive speed of not less than 15 frames per second, *micro-geometry* indicates the ability of that method to represent the fine variations in height of individual points on the surface, memory usage indicates the amount of storage memory required to represent a sample relative to a standard texturemap. *Self-occlusion* and *self-shadowing* indicate the ability of a *3D surface representation* to obscure itself from the line of sight of an observer or light source. Practical acquisition methods include the ability of the *3D surface representation* to be acquired using current image processing technology.

Method	**Real-Time**	**Practical Acquisition Methods**	**Memory Usage**	**Micro-geometry**	**Self Occlusion / Self Shadowing**
General and Scattering Functions	No	No	x3+	No	No
BSSRDF	Yes	No	x3+	No	No
BTF/SLF/SRF	Yes	Yes	x64	Yes	No
BRDF/DSRF	Yes	Yes	x3+	No	No
Point-Set Surfaces	No	Yes	x1	Yes	Yes
PTM	Yes	Yes	x3/x6	Yes	No
Texture-mapping	**Yes**	**Yes**	**x1**	**No**	**No**
Bump-mapping	**Yes**	**Yes**	**x2**	**Yes**	**No**
Relief mapping	**Yes**	**Yes**	**x2**	**Yes**	**Yes**

Table 1: Summary of the candidate 3D surface representations

2.2 Review of Real-Time 3D Visualisation methods

In this section, we survey publications related to the interactive visualisation of *3D surface representations*, with a particular focus on the use of *real-time* lighting and shadowing techniques to relight *3D surface representations* acquired using *photometric stereo*. As mentioned previously, the purpose of the 3D *surface representation* is to represent the *micro-structure* of a textile in a common data format that allows for both *texture retrieval feature vectors* and *real-time 3D visualisation* of the textile samples to be performed. While the *3D surface representation* is used to represent the *micro-structure*, the *macro-structure* or curvature is represented using 3D geometry. The 3D

geometry normally consists of vertices arranged in a mesh to form triangles or polygons or the *3D geometric object*. Having a *real-time 3D visualisation* system is an essential part of a modern feature-rich *texture retrieval* system, as it assists the user in immediately seeing which textile samples they have selected. For a fully function interactive *texture retrieval* application, the user would be able to select a textile sample along with which particular attributes they preferred and disliked. Such attributes would include colour, dominant direction and pattern. We begin our review of *3D visualisation* methods by first providing a review of shadowing methods.

2.2.1 Selection of the lighting and shadowing method

As mentioned in the previous section, *programmable graphics accelerator boards* offer an unprecedented level of performance. One of the most useful features possible with *programmable graphics accelerator boards* is the ability to implement photorealistc lighting effects with shadows. As one of the goals of this thesis is to render textiles as realistically as possible, we wish to visualize the geometry with visible shadows. To achieve this goal, we perform a survey on all related literature on the use of *shadow mapping* techniques. We have three criteria for this technique. These are as follows:

- Operating in *real-time*
- Does not require any preprocessing of geometry – no static light sources
- Can take advantage of hardware acceleration

Real-time operation is essential if the user is to be able to use the application effectively. Preprocessing of geometry is not desirable as this makes the assumption that all light sources are stationary, and can take many hours with large geometric objects. Hardware acceleration is desirable as this allows for the rendering speed to be increased.

Because of the active interest in the field of photorealistic rendering, publications describing new shadow rendering techniques have been published on a regular basis, with surveys on the state-of-the-art being published every four years or so. The most recent survey on shadow rendering techniques was by Hasenfratz who identified seven general methods of rendering shadows with *3D geometric models* [Hasenfratz2003]. We list these methods below:

- Shadow volume
- Radiosity/Discontinuity meshing
- Ray-tracing
- Scan line algorithms
- Subdivision methods
- Shadow mapping
- Shadow field

Each of these shadowing methods can be implemented in either software or custom hardware. However, some methods are more suited towards working with large environment scenes which extend to many times the size of the view frustum, while others are more suited to small geometric objects which fit entirely the view frustum.

Shadow volumes and radiosity/discontinuity meshing are both examples of environmental techniques in that they process the entire scene as a whole and will not work for small convex geometric objects. Shadow volumes require the generation of shadow planes from the intersection of polygon edges and planes to generate distinct shadow boundary edges, while radiosity and discontinuity meshes use a finer mesh to model the gradual change between the penumbral and umbral areas of a scene. We consider these methods due to their ability to handle scenes of large complexity.

Ray-tracing, scan line algorithms, subdivision methods and *shadow mapping* methods are all forms of view frustum techniques, due to the fast that the calculations performed will vary according to the location of the observer viewpoint. However, these techniques differ in that ray-tracing scans the entire framebuffer on a pixel-by-pixel basis, while scan line algorithms scan individual geometric objects on a pixel-by-pixel basis, subdivision methods subdivide the polygonal geometry visible within the view frustum until the depth order is unambiguous, and the *shadow mapping* method relies on two views of the scene being rendered; one from each light source, and one from the observers viewpoint. We choose to investigate these methods because of their ability to operate with polygonal geometry.

The Shadow field method differs from all the other techniques in that it defines a local spherical shadow region around each object rather than manipulating polygons or projecting textures. We choose to investigate this method because of its ability to work with complex scenes.

The introduction of *programmable graphics acceleration boards* has allowed each of these methods such as shadow volumes and *shadow mapping* to operate in *real-time* (at least 15 frames per second). The introduction of *programmable graphics accelerator boards* has also allowed researchers to implement methods such as ray-tracing, radiosity and discontinuity meshes at *real-time* speeds for small geometric models and close to *real-time* for large scenes. We now perform a detailed review of each of these methods:

2.2.2 Shadow volumes

Shadow volumes involve the creation of infinite half volumes by extruding the silhouette edges of each occluding objects away from the direction of the light source, and converting each extruded edge into a shadow polygon [Crow1977] [Nishita1983] [Fuchs1985]. This has the benefits that determining whether a single point in the scene is in a shadow region or not is achieved simply by counting the number of shadow planes that are crossed between the viewpoint of the camera and the current point. Forward facing planes increase the count, while backward facing planes decrease the count. A point that has a non-zero count is considered to be in a shadow region. However, one problem with this method involves the situation when the viewpoint is already in a shadowed region, as this will offset the count by one. The advantages of this method are that it can take advantage of hardware acceleration and operate in *real-time* [Heidmann1991] [Laine2005]. Unfortunately, the disadvantages of this method are that it requires pre-calculation of all shadow planes using static light sources. As the criteria for this thesis is to implement shadow rendering using dynamic light sources under user control, we do not consider this method suitable for the needs of this thesis.

Figure 14: Scene rendered using shadow volumes

(from [Laine2005])

2.2.3 Radiosity/Discontinuity meshing

Radiosity methods involve breaking up a scene into separate regions, and modeling the amount of light reflected and absorbed between the different regions. The benefits of this method are that the resolution of the radiosity mesh can be made as small or as large as required by the user. In the past, radiosity calculations could only be implemented in software. However in recent years, it has become possible to implement radiosity calculations using *programmable graphics accelerator boards* [Coombe2004]. In his paper Coombe describes how scenes comprised of up to ten thousand elements could be rendered in less than one second using a radiosity rendering algorithm running on a GPU. For a scene consisting of over one million elements, such a scene would take 86 seconds to render. We present a sample image from this paper in (Figure 15). The disadvantages of this method are that while it can be implemented using GPU hardware, it cannot run at *real-time* frame rates. As our criteria are that our system should be able to run in real time, we find that this method is not suitable for our needs.

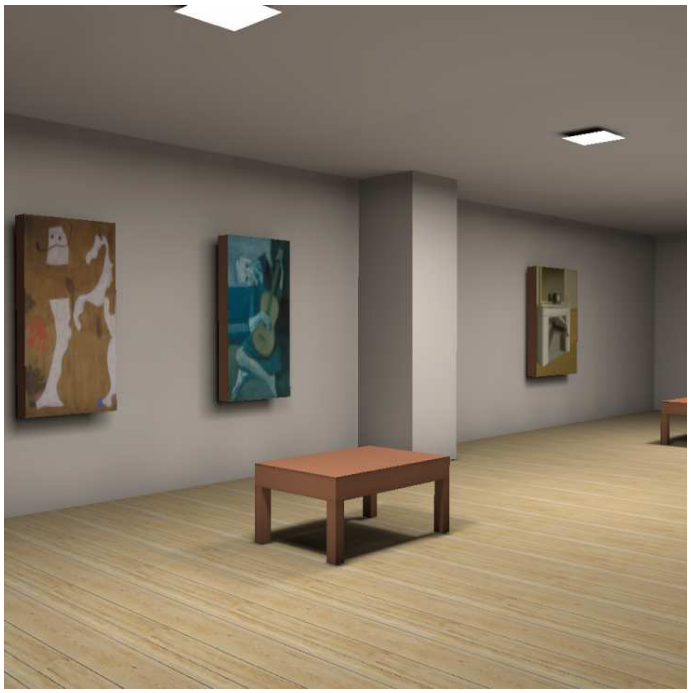


Figure 15: Scene rendered using radiosity calculations

(from [Coombe2004])

2.2.4 Ray-tracing methods

Ray-tracing methods involve scanning the viewing region of the camera pixel by pixel and performing one or more ray-object intersection tests for each pixel [Kay1979] [Kay1986]. The benefits of using ray-tracing are that complex reflections and refractions can be generated, but at the cost of increased processing time per pixel. For those objects that intersect the ray, the point of intersection is calculated and used to generate texture coordinates, and new rays to model reflection, refraction and shadow calculations. Determining whether a region is in shadow or not, is achieved by performing a ray-object intersection test on the line between the source of illumination and the point on the surface. If any object intersects this line, then the point is in shadow; otherwise, it is in full view of the light source. The advantages of this method are that it offers the highest level of photorealism (Figure 16). The disadvantages of this method are that it does not run in *real-time*; ray-tracing a complex scene on a single processor can take several minutes if not hours. For this reason, we do not consider this method suitable for the needs of this thesis.

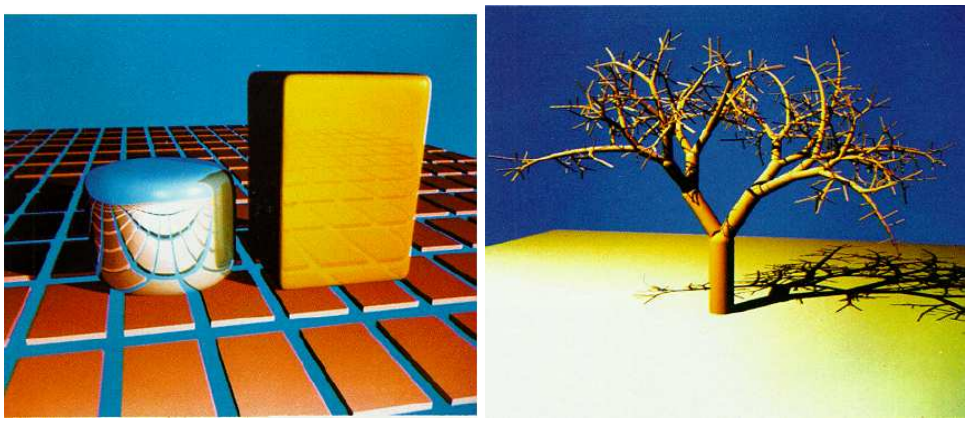


Figure 16: Raytraced scenes with shadows

(from [Kay1986])

2.2.5 Scan line algorithms

Scan line algorithms involve using software to render each individual *3D geometric model* pixel by pixel within each scan-line. The benefits of using software rendering are that it is extremely flexible in how lighting and shadowing effects may be implemented especially when combined with the Watkins hidden surface algorithm. Andonian and Toida describe a method of calculating the shadow projections from multiple light sources using *perspective projections* [Andonian1978]. However, the disadvantages of using software *rendering methods* are that rendering is not in *real-time*. With the introduction of *graphics accelerator boards*, software implementations of this method has become less attractive, as it is more efficient to tessellate *parametric surfaces* into triangles, and use hardware accelerated triangle rasterisation instead. The software implementation of method has the disadvantage of not operating in *real-time* or supporting hardware acceleration. As the goals of this thesis are to implement *real-time rendering*, this method does not match the criteria specified by this thesis.

2.2.6 Subdivision methods

Subdivision methods involve the use of polygon clipping to subdivide the current view of the scene into smaller and smaller squares until the depth order of the remaining polygons can be determined. This method has the advantage that a scene need only be specified in terms of light sources and N-sided polygons, with the subdivision method

generating the resulting rendered geometry. Weiler and Atherton describe how a scene may be rendered by clipping the visible polygons by the silhouette of each other until the depth order is determined [Weiler1977]. The method requires that all polygons are depth sorted relative to the observer, then sorted relative to the nearest polygon. Any pairs of polygons which intersect in terms of nearest and furtherest vertices are clipped relative to each other. This depth sorting algorithm could also be adapted to the calculation of shadow regions by replacing the observer viewpoint with the light source location not (Figure 18). The advantages of this method are that it generates object accurate shadows. The disadvantages are that clipping takes a considerable amount of times, especially with scenes composed of large amounts of geometry. Chin proposed a method of using BSP trees to speed up the process of the generation of shadow regions with the enhancement that both penumral and umbral regions of shadows could be generate [Chin1992]. The advantages of this method are the improved accuracy in shadow generation. However, there are several disadvantages to this method. The first is that it only works with static light sources, while the second is that the generation of shadow regions stills requires pre-computation. Other disadvantages are that this method does not operate in *real-time*, nor can it take advantage of hardware acceleration. Since being able to operate in *real-time* is one of the criteria of our visualisation system, we do not consider this method suitable for the needs of this thesis.

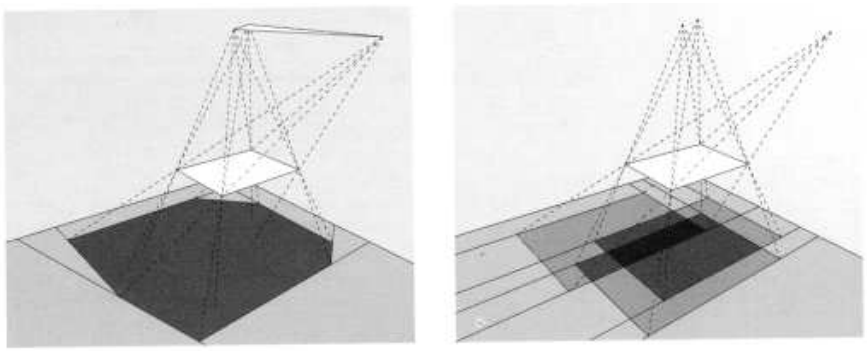


Figure 17: Subdivision of scene into umbral and penumbral shadows

(from [Chin1992])

Figure 18: Scene rendered using subdivision methods

(from [Weiler1977])

2.2.7 Shadow mapping

Shadow-mapping is another method of determing shadow regions [Williams1978]. In order to eliminate the need for complex data structures, the *shadow-mapping* method involves two rendering stages. In the first stage, the application renders a view of the scene as seen by the lightsource (Figure 20), with the depth map information kept for use with the second stage (the *shadowmap*). In the second stage, the application renders a view of the scene from the viewpoint of the observer (Figure 19), with the depth-value of each pixel transformed into the coordinate system of the lightsource and compared against those of lightmap, giving a Boolean result (Figure 21). Depth values further away than than the value in the depth map indicate areas that are in shadow, while those closer than the value in the depth map are in view of the light source (Figure 22). The introduction of depth textures and render-to-texture options in graphics accelerator boards has also made *shadow mapping* another one of the most popular methods.

With the introduction of *programmable graphics accelerator boards*, it has also become possible to use the *shadow-mapping* technique to render scenes with either hard or soft shadows [Valient2005] [Atty2006]. Hard shadows consist exclusively of an unbral region and have an abrupt change from light to dark, while soft shadows consist of a shadow region consisting of both penumbral and umbral regions, thus producing a

blended region of illuminated and shadowed regions. A psychophysical study performed by Wanger [Wanger1992], came to the conclusion that: “Computationally cheaper hard shadow generation techniques are adequate and in fact may actually be more beneficial than more expensive soft shadow techniques”.

The advantages of this method are that it can operate in *real-time* through the use of hardware acceleration and does not require any preprocessing of geometry. Another advantage of this method is that it is possible to implement both soft shadows simply by calculated a weighted sum of multiple sample points within the *shadowmap*. Because this method does not require any precomputation, dynamic and multiple light sources are easy to implement. As this conforms to our criteria, we consider this method suitable for the objectives of this thesis, and worth further investigation.

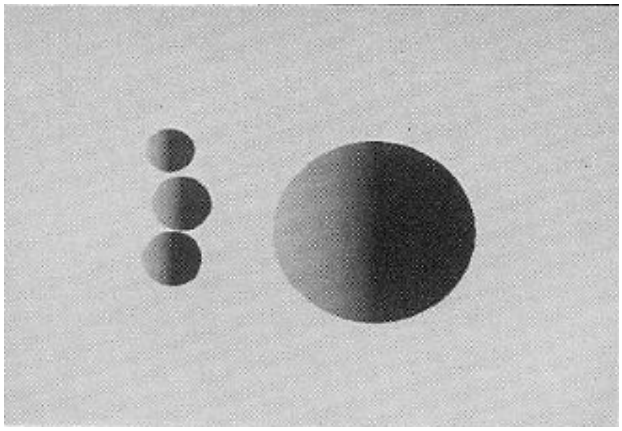


Figure 19: Observer view of shadow mapping scene

(from [Williams1978])

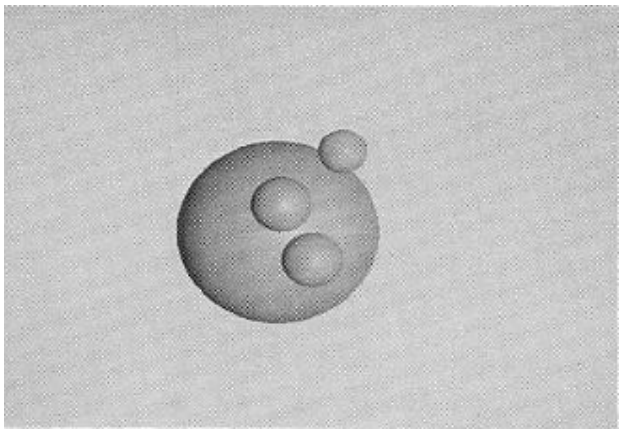


Figure 20: Light source view of shadow mapping scene

(from [Williams1978])

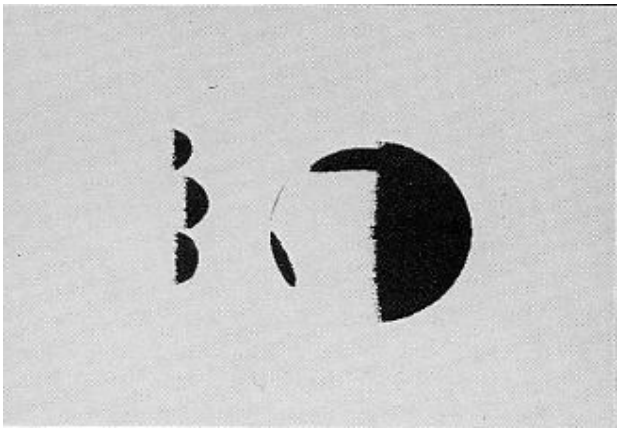


Figure 21: Shadowmap of scene

(from [Williams1978])

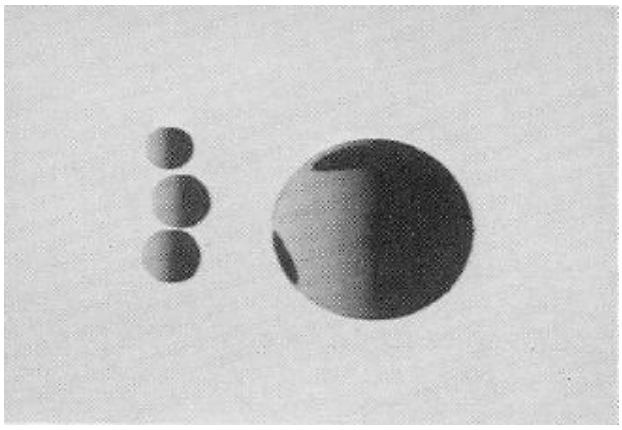


Figure 22: Observer view of shadow mapped scene combined with shadows

(from [Williams1978])

2.2.8 Shadow fields

Shadow fields are one of the most recent methods of shadow generation [Zhou2005] [Ren2006] developed to support the rendering of soft shadows. This method represents the light intensity field surrounding an object as a concentric set of thirty-two low-resolution cube-maps (32x32 squares x 6 sides) (Figure 23). Fourth or fifth order *spherical harmonics* are used to compress this data so that it may be used using a *programmable graphics accelerator board*. The memory requirements for this method take range from 15 Mbytes to 500 Mbytes depending upon the geometry of the model. Frame render rates range from 0.1 to 18 frames per second. The advantages of this method are that it can operate in real time and render soft shadows using dynamic light sources. However, the disadvantages of this method are that precomputation of the shadow fields can take up to two hours. As the criteria for this thesis is to avoid the need for pre-computation of data, we find that this method is not suitable for our needs.

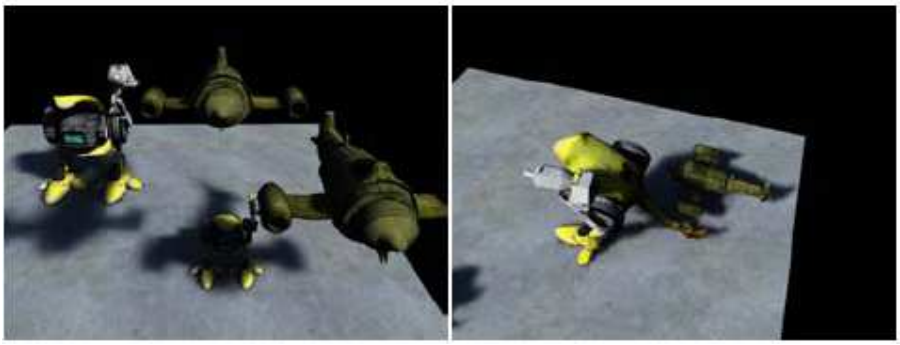


Figure 23: Soft shadows generated using shadow fields

(from [Zhou2005])

2.2.9 Summary

We have reviewed the literature relating to shadow rendering techniques and have made the following observations. We can classify each of these methods into one of three types of shadowing technique; environmental, view-frustum and local object.

Environment based shadow-mapping techniques

In environmental techniques, the 3D geometry consists of a representation of a complex topological structure with floors, walls, ceilings, steps and other architectural features all represented using polygons. In our survey, there are two such methods suitable for this type of data; shadow volumes and radiosity/discontinuity meshing.

With the Shadow volume method, shadow volumes are pre-calculated from projecting rays from each point light source to the vertices of the geometry and extending these until each ray intersects another polygon. For each edge of a polygon, a plane of the shadow volume is generated. As a result, this method requires that both the location of every light source and polygon remain static. Thus, for this reason, this method is not suitable for our needs.

The Radiosity/Discontinuity meshing method supports dynamic light sources through the statistical calculation of the redistribution of light energy due to reflection. While radiosity scenes consisting of less than 10,000 elements can be rendered using a GPU in less than 1 second, radiosity scenes consisting of over 1,000,000 elements take over 80 seconds to render using a GPU If converted to a radiosity scene, a single textile sample at a resolution of 512x512 would consist of over 520,000 elements, thus requiring a rendering time far in excess of 15 frames per second. For this reason, we do not consider this method suitable for our needs.

View-frustum shadow-mapping techniques

With view-frustum techniques, the view of the scene is rendered pixel by pixel using a suitable algorithm for primitive geometry shapes such as triangles, spheres and N-sided polygons. For each rendered pixel, a illumination test is performed to see if the associated point in three dimensional space is illuminated by each light source or not. In our survey, there are four such methods; scan-line algorithms, subdivision methods, shadow-mapping and ray-tracing.

With scan-line algorithms, the basic geometry primitives are triangles and N-sides polygons. However, as this method requires a software implementation, it cannot

benefit from hardware acceleration and thus does not operate in *real-time*. Thus, we do not consider this method suitable for further investigation.

Subdivision methods render the current view of the scene by subdividing the scene until the order of the polygons is consistent. Shadow calculations are implemented by generating two ordered polygon lists, the first being the scene visible from the light source, and the second the scene visible from the camera. Whether a polygon is visible or not from the light source determines whether or not it is in shadow. As this method requires a software implementation, it cannot benefit from hardware acceleration and thus does not operate in *real-time*. Consequently, we not do consider subdivision methods suitable for our needs.

The ray-tracing method supports dynamic light sources, but does not benefit from hardware acceleration and thus does not operate in *real-time*. This we do not consider this method to be suitable for our needs.

The *shadow-mapping* method renders the shadow from a single light source in two passes. In the first pass, the *depthmap* of the scene visible from the viewpoint of the lightsource is calculated. In the second pass, the scene is rendered as normal from the viewpoint of the camera, with a depth comparision test being performed on each rendered pixel. Because the depth comparison tests can be performed in parallel this method benefits from hardware and operates in *real-time*. Thus we consider this method suitable for our needs. Other advantages of this method are that it is possible to implement soft-shadows by calculating a weighted sum of multiple sample points within *the depthmap*, and that this method can be extended to multiple light sources.

Local object methods

Local object methods differ from the previous two methods in that they represent the projected shadow of a small dynamic 3D geometric object rather than an entire scene. In our survey, only the shadow-field method belongs in this category.

The shadow-field method represents the shadow of a *3D geometric object* by using a set of small cube maps (32x32x32) to represent the light intensity in the space surrounding

the object, and then compressing this data using fourth order *spherical harmonics* However, due to this representation, this method does not operate in real-time with complex geometry. For this reason, we do not consider this method suitable for our needs.

We present a table summarizing the properties of each of these methods below (Table 2). In this table, dynamic light sources indicate the ability of light sources to be moved around at run time by the user. Hardware accelerated indicates the ability of the shadow method to take advantage of *programmable graphics acceleration hardware*. *Real-time* indicates the ability of the shadow method to run at 15 frames per second or faster.

Shadow method	**Dynamic light sources**	**Technique**	**Hardware accelerated**	**Real-Time**
Shadow volume	No	Environment	Yes (Stencil buffer)	Yes
Radiosity/ Discontinuity meshing	Yes	Environment	No	No
Scan-line algorithms	Yes	View-frustum	No	No
Subdivision methods	Yes	View-frustum	No	No
Shadow map	Yes	View-frustum	Yes (Shadowmap)	Yes
Ray-tracing	Yes	View-frustum	No	No
Shadow field	Yes	Local object	No	Yes (simple objects)

Table 2: Summary of basic shadow methods

In this section we have identified seven candidate shadow generation methods, and identified *shadow-mapping* as the most suitable for our needs. The *shadow-mapping* method is ideally suited to implementation on a *programmable graphics accelerator board* due to the built-in *shadow-mapping* hardware extensions which support 3D perspective transformation and depth-map comparision tests. Consequently, this method has the advantage of running in *real-time* and not requiring any pre-computation of complex data structures. Based upon the conclusions by Wanger, we choose not to

implement soft shadows but just to implement hard shadows instead, although it would be trivial to implement soft shadows in the future if required. While the shadow-mapping method solves the problem of inter-object and convex-object shadow generation, it does not solve the problem of shadow generation of the *micro-geometry* of the textile sample. This problem can only be resolved through the use of *relief-mapping*. Combining these two methods together is still an issue that must be resolved.

2.3 Review of Rotation invariant texture retrieval features

As the goal of this thesis is to implement an information retrieval system based upon *rotation invariant texture retrieval*, it is necessary to have a method of indexing and searching through database entries. To achieve this goal with *texture image data*, it is necessary to have a simple data field that can be rapidly compared with other entries. We refer to this as the *feature vector.* Early *texture databases* could only define the *feature vector* as a short text description due to the limited storage space available. While easy to use, this method required that every image had to have a detailed description entered manually in order for the database search requests to be useful. One solution to this problem is to have the *feature vectors* generated automatically from the original *texture database* images, and then search for a match with the *feature vector* derived from the target image selected by the user. This reduces the problem of searching *texture databases* down to finding a useful and economical way of generating and comparing *feature vectors*. In the follow section we define our criteria for the selection and evaluation of the selected *texture retrieval* methods.

2.3.1 Selection of the texture retrieval method

In order to identify the most suitable methods to evaluate, we perform a survey of existing texture feature classification and retrieval methods and then compare them against the following criteria:

- *Must select textures based upon similarity*

  We would like the user to be able to select a target texture and have the retrieval system find those textures in the database that are as similar as possible to the target texture in terms of colour and *micro-geometry*.

- *Must support colour albedo and surface representation images*

  As one of the objectives of this thesis is to implement virtual reality textile catalogues, it is essential that the retrieval methods must be able to work with true-colour images (at least eight bits for each of the red, green and blue colour channels).

- *Must be rotation invariant*

  Rotation invariance is the ability of a *texture retrieval* system to match two similar images regardless of their rotational orientation. Whenever more than one *photometric image* of a textile sample is made, each image will always have a unique rotation, due to slight differences in the position of each texture sample relative to the recording device. Consequently, this will distort any *feature vector* derived from this image. We describe *texture retrieval* methods that are able to overcome this problem as being *rotation invariant*.

- *Must have efficient memory usage*

  An uncompressed 512 x 512 pixel true-colour image with 16-bits of data per pixel for each of the red, green and blue colour channels will occupy 1.5 Mbytes of memory, while a image with 8-bits per colour channel will occupy 768 Kbytes of memory and a JPEG compressed image will still occupy more than 400 Kbytes of memory. Even with such high levels of data compression, these amounts of memory used are still far too high for *texture retrieval* to operate interactively. Consequently, any *feature vector* derived from a *texture image* must be considerably smaller than this. Given that the *texture database* may be located on a separate server, and accessible only via a dial-up connection, an upper limit of 8 Kbytes per *feature vector* is considered to be efficient memory usage.

- *Must be able to encode perioidic information*

  By their nature, woven textiles have a periodic pattern generated from various parameters such as the size of thread, the weaving pattern used and

the density of the thread pattern. Any of these parameters can be measured using 2nd order statistics such as Amplitude, Magnitude and Power spectra.

In her thesis, Taylor [Taylor2003] identified that power spectrum methods were ideally suited to identifying the characteristics of woven fabrics, due to the fixed distance and axis alignment of the threads of each woven fabric creating periodic patterns.

In his PhD Thesis, "Rotation Invariant Classification of 3D Surface Texture Using Photometric Stereo" [Wu2003], Wu performs a survey of methods used to define *texture features*.

Wu follows the *texture classification* scheme as defined by Tuceryan and Jain [Tuceryan1993] and assigned each *texture classification* method to one of four branches; statistical methods, geometrical methods, model-based methods and signal processing methods. We present this classification as a hierarchical chart in (Figure 24). There are two main types of signal processing method; these are linear and non-linear methods. Linear methods involve either a convolution in the *spatial domain* or a Fourier transformation into the *frequency domain*. Non-linear methods are based upon other relationships between pixels. These can either be based upon geometry, statistics, or mathematical models. These form four main branches; statistical based methods, geometrical based methods, model based methods and signal processing based methods.

These techniques have, in general been applied to photometric grey scale or colour information. Only a few researchers have applied them to *micro-geometry* data [Wu2003] [McGunnigle1997] [McGunnigle1998] [McGunnigle1999]. A particular issue here is that data such as *Surface normal* fields often contain directional artifacts which can cause problems for *rotation invariant* systems.

We now choose to investigate candidate methods in each of these branches due to their ability to perform *texture classification* bearing the above issues in mind. We begin our investigation by examining statistical based methods.

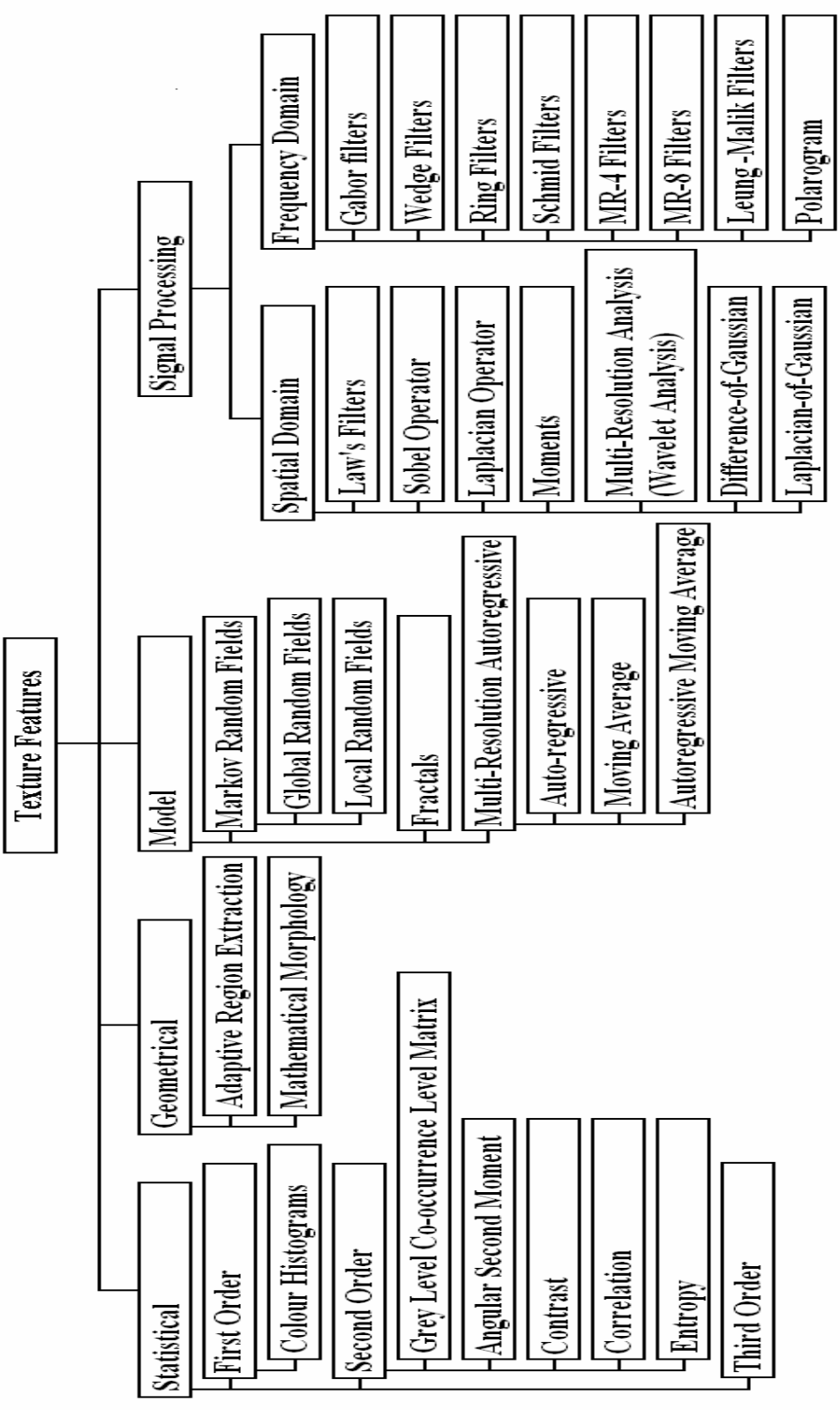


Figure 24: Hierarchy of texture classification methods

2.3.2 Statistical methods

Statistical methods involve the analysis of the spatial distribution of gray values, or in our case, micro-geometry and colour data, by computing local features for every point in the image, and using the resulting distribution of the local features to generate a set of statistics [Julesz1981] [Julesz1983]. Depending upon the number of pixels used for analysis, the resulting set of statistics is defined as being either first-order, second-order or third order.

2.3.3 First order statistics

First order statistics calculate a measure of difference in overall brightness. These include the calculation of the mean average or *histogram* of the *texture image data*. Second order statistics calculate differences in granularity and slope. These include the gray-level co-occurrence matrix and the the statistics derived from this matrix. Third order statistics are used to identify deviations from the standard Gaussian distribution. These include the 3rd order cumulant and 3rd order bispectrum.

Swain and Ballard describe a method of performing similarity matching between images using *colour histograms* using a method they call *histogram* intersection [Swain1991]. In their paper, they determine that 2048 bins (16x16x8) are required to implement a *colour histogram* suitable for matching. The *colour histogram* transforms each RGB colour pixel into an index within the corresponding *histogram*, and increments the associated bin. Performing a similarity match between two *colour histograms* is achieved through the comparison of individual pairs of bins from each of the two *colour histograms*. Typical functions include the sum of absolute differences or the sum of the minimum values. To allow for the comparison of images of different dimensions, both *histograms* and *colour histograms* are normalized by dividing each bin by the total number of pixels in the image, thus giving a fractional value.

The advantages to this method were that *histograms* can be rapidly calculated without requiring a transformation in the *frequency domain*, and are unaffected by changes in orientation, viewing position or even shape. The main disadvantage of the *color histogram* is that it is sensitive to changes in lighting conditions; either when the direction of or colour of illumination changed.

Funt proposed a solution to this problem through the use of *colour ratio histograms*, where instead of using the colour values of the image directly, the derivative (Laplacian or first directional derivatives) of the "logarithm of the colours" would be used to index the *histogram* bins [Funt1991]. This method had the advantage of removing any variation due to illumination direction but had the disadvantage of requiring a convolution filter to be applied to the entire image before the generation of the *histogram*.

However, this is not a concern for us as we require the image to be captured under controlled condition required to obtain the *micro-geometry*.

Stricker proposed an alternative solution through the use of "*boundary histograms*" [Stricker1992]. Using this technique, the entire image is processed by a colour constancy algorithm in order to correct for any variations in colour of the illumination light source, then processed another time to convert the color space into a discrete colour space before 2x2 blocks of pixels are analyzed. The length of the boundary edge is estimated from analysis of the color distances from the four pixels within that block. The resulting edge length is then used to increment the associated *histogram* bin. The advantage of this method are the the *boundary histograms* are compact in size, are not affected by noise. The disadvantages of this method are that the length of boundary edges are only estimated to within a 5% error tolerance, and are thus not completely *rotation invariant*. We present a table comparing the features of these different *histogram* methods below (Table 3):

Method	**Rotation invariant**	**Pre-processing**
Colour histograms [Swain1991]	Yes	No
Colour ratio histograms [Funt1991]	Yes	Yes
Boundary histograms [Stricker1992]	No	Yes

Table 3: Comparison of histogram methods

From this table it can be seen that both *colour histograms* and *colour ratio histograms* are *rotation invariant*. However, it can also be seen that while all of these statistical

methods have the advantage of not requiring transformation into the *frequency domain*, methods such as the *colour ratio histogram* and *boundary histograms* require pre-processing in terms of a convolution filter. Another disadvantage with these methods is the high dimensionality of the *histogram* data. As each *histogram* can have up to 256 bins, each *histogram* can be considered to be a 256 dimension vector. However, this disadvantage is outweighed by the speed of calculation. While regular photographs would have the disadvantage of having different lighting conditions, in our thesis, the lighting conditions are under our complete control and are identical for every photometric image taken. Thus, for this thesis we choose to use *colour histograms* as described by Swain and Ballard.

2.3.4 Second order statistics

Second order statistical methods include grey-level co-occurrence matrices [Davis1981] and the statistics derived from this matrix; the entropy, energy (angular second moment or Haralick second moment), contrast, homogeneity, mean, variance, correlation, maximum probability, inverse difference moment and cluster tendancy [Clausi2002]. The advantages of this method are that it does not require a transformation into the *frequency domain* in order to generate the statistics.

2.3.5 Geometrical methods

Geometrical methods attempt to model the appearance of a texture by reducing the texture down to a combination of fundamental geometric primitives and placement rules. This transforms the problem of *texture classification* into the identification of the fundamental geometric primitives that form each texture or *texture analysis*. There are two geometrical methods that have been developed; mathematical morphology and adaptive region extraction. However, in previous research, these have in general been applied to photometric rather than geometric data.

2.3.6 Mathematical morphology

Mathematical morphology is a technique originally developed by Matheron and Serra in order to perform *texture analysis* on ore grades still underground [Matheron1975] [Serra1973]. This method involves the application of set theory to geometry. Operations supported by mathematical morphology include “erosion” and “dilation” [Ledda2002].

Through these two operations it is possible to determine the skeleton and convex hull of the components of an image. The advantage of this method is that it does not require images to be transformed into the *frequency domain*, and that similarity can be measured through pattern spectrum analysis. However, the disadvantage of this method is that it is intended for use with images with distinct silhouette edges rather than continuous patterns. For this reason, this method does not conform to our criteria for candidate methods of the classification of textile textures.

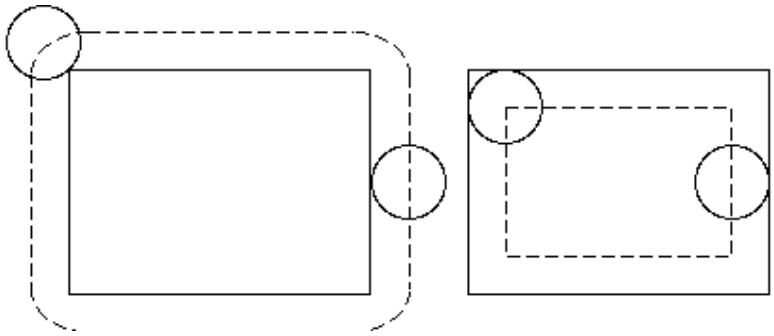


Figure 25: Erosion and Dilation in mathematical morphology

(from [Ledda2002])

2.3.7 Adaptive Region Extraction

Adaptive Region Extraction is based upon the extraction of contours from an analysis of pixel intensity values [Hong1980]. Hong describes a method of implementing such a system in four stages. The first stage is to apply an edge detection operators to the image to identify candidate edges. The second stage applies a thresholding filter to eliminate noise while the third stage is to apply non-maximum suppression to eliminate redundant responses to a single boundary. The fourth stage is the application of eight 3x3 convolution filters to identify edges in the horizontal, vertical and diagonal orientations. Classification of the resulting texture was then achieved through the calculation of six first-order statistics which included the area of each region, the perimeter of each region, the dispersedness of the region, the elongatedness of each region, the eccentricity of each region, the major axis direction and the average gray level.

The advantage of this method is that it does not require a transformation into the *frequency domain* and that the statistics generated are *rotation invariant*. However, the disadvantage of this method is that it has been designed to work with edge thresholding with grey-scale data to identify areas of differing intensity rather than colour textures or geometric data, and so we do not consider it any further.

Figure 26: Texture classification using adaptive feature extraction

(from [Hong1980])

2.3.8 Model-based methods

Model-based methods attempt to mimic the process that generated the original texture through the use of stochastic models, and can be subdivided into three main methods:

- Markov Random Fields
- Fractals
- Multi-Resolution Auto-Regressive Features

2.3.9 Markov Random Fields

Markov Random Fields are a way of generating random patterns through the use of Gibbs random field models. The Markov Random Field assumes that there is a conditional relationship between the grey-scale intensity of one pixel and the immediate surrounding neighbourhood of pixels. Such models are described as local random fields. In the case when it is necessary for each pixel to have an influence over the entire *texture images* the model is described as a global random field. Chellappa describes the Gaussian Markov Random Field (GMRF) in which the Gaussian distribution is used to define the probability function for a small region around each pixel [Chellappa1993].

Because of this dependence on pixel intensity values, analysis of colour images requires conversion to grey scale before processing. Another disadvantage with the GMRF is that the dependence on the Guassian distribution constrains the *texture analysis* to the highest frequencies within the texture. Haindl and Vácha describe a solution to the use of this problem through the use of multiscale decomposition using the Gaussian pyramid, a sequence of images in which each image is downsampled from its predecessor and has a low-pass filter applied. This allows a *feature vector* to be generated from the set of GMRF values derived from the Gaussian pyramid. However, GMRF techniques encode only 2^{nd} order information, and therefore have no more discrimination power over linear filtering while being more difficult to understand and design.

2.3.10 Fractals

Fractals have traditionally been a method of classifying the complexity of textures and geometry by deriving a ratio or the "fractal dimension" between the perimeter and the surface area of a texture or the surface area and volume of a three-dimensional object [Mandelbrot1983]. However, this single parameter simply defines the linear frequency roll-off factor of the power spectrum and is incapable of modeling the periodic nature of weaves, and thus we do not consider it any further.

2.3.11 Multi-Resolution Auto-Regressive Features

Sarkar, Sharma and Sonak described a method of using the two-dimensional autoregressive moving average (ARMA) model to perfom *texture analysis* on the Brodatz database [Sarkar1997]. In their paper, they describe a method of solving the set of 2D transcendental functions of the autoregressive components through the use of singular value decomposition (SVD) and factorization techniques, to generate a *feature vector* of twenty-four dimensions. This method exploits power spectrum data but in the *spatial domain*. When considering a complete filter bank, this method is more than likely to be computationally more expensive than *frequency domain* techniques.

2.3.12 Signal processing methods

Signal processing methods can be implemented in the *spatial domain* and the *frequency domain*. Signal processing methods operating with *spatial domain* data operate on pixel data directly, while signal processing methods operating with *frequency domain* data operate upon the data generated from a Fourier transformation. Linear methods can be implemented in either domain with the choice being made on coding pragmatics and computational efficiency.

2.3.13 Spatial domain methods

These include edge detection filters such as the Law's filter (linear spatial convolution), the Sobel operator, the Laplacian operator, Moments, Wavelet Analysis and the Difference-of-Gaussian operator. Each method works by performing a mathematical function on each pixel and the surrounding block of neighbouring pixels around it, typically for a block size of 3x3 or larger. *Texture classification* using *spatial domain* methods simply involves applying the filter to the image and calculating the rectified sum of all pixel intensity values. For larger block sizes, *frequency domain* methods are more efficient in terms of processing time. The advantages of using *spatial domain* methods are that they are easy to calculate and more efficient for small masks. As we are considering linear techniques that can be implemented in either domain, and as the *frequency domain* provides easier methods for designing banks of filters we will consider the *frequency domain* characteristics of these filters.

2.3.14 Frequency domain methods

With *frequency domain* methods, the *spatial domain texture image* is transformed by the discrete FFT as the first stage of calculating the *feature vector*. Calculation of the *feature vector* using a set of filters or *filter bank* involves multiplying the *frequency domain* image of each filter in the *filter bank* and then applying the inverse discrete FFT to return the combined image back into the *spatial domain*. The coefficient value for the filter is then calculated by summing together all the rectified pixel values in the resulting image. All *filter banks* have the advantage of being able to be used with colour *texture data* as well as grey scale *texture data*. Another advantage of *frequency domain filter banks* is that the *feature vectors* are either naturally *rotation invariant*, or can be

made to be through the use of index offsetting when comparing *feature vectors*. The main disadvantage to the use of *frequency domain* method is that computation of the Fast Fourier Transform and its inverse which can be inefficient for small masks. However, as they conform to our criteria of being able to exploit periodic data and be used with colour *texture data* and are *rotation invariant*, we consider them to be of use to this thesis. Randen performed a survey of both *spatial domain* and *frequency domain* methods for *texture classification* [Randen1999]. In his paper, Randen identified the following classes of *frequency domain filter banks*:

- Ring and Wedge filter banks
- Gabor filter bank

In addition to these we consider the following *filter banks*:

- Schmid filter bank
- Leung-Malik filter bank
- MR4 and MR8 filter banks
- The Polarogram

2.3.15 Ring and Wedge filter banks

Coggins and Jain identified both the *ring filter bank* and the *wedge filter banks* as being suitable for the task of *texture classification* [Coggins1985]. In their system, seven dyadically spaced *ring filters* and four *wedge filters* for *texture classification* (Figure 27). This system forms a *feature vector* of eleven elements for gray scale data and thirty-three for red, green and blue colour textures. While the *ring filters* are *rotation invariant* and are sensitive to frequency only, the *wedge filters* are not *rotation invariant* and are sensitive to direction only. However as a *wedge filter bank* can be made to be *rotation invariant* through the use of index offseting, the *combined filter bank* can be made to be *rotation invariant*. As this method conforms to both our critieria that the *texture retrieval* method should be *rotation invariant* and support the use of colour *texture data*, we consider both the *ring filter bank* and *wedge filter bank* to be candidate methods for *rotation invariant texture retrieval*.

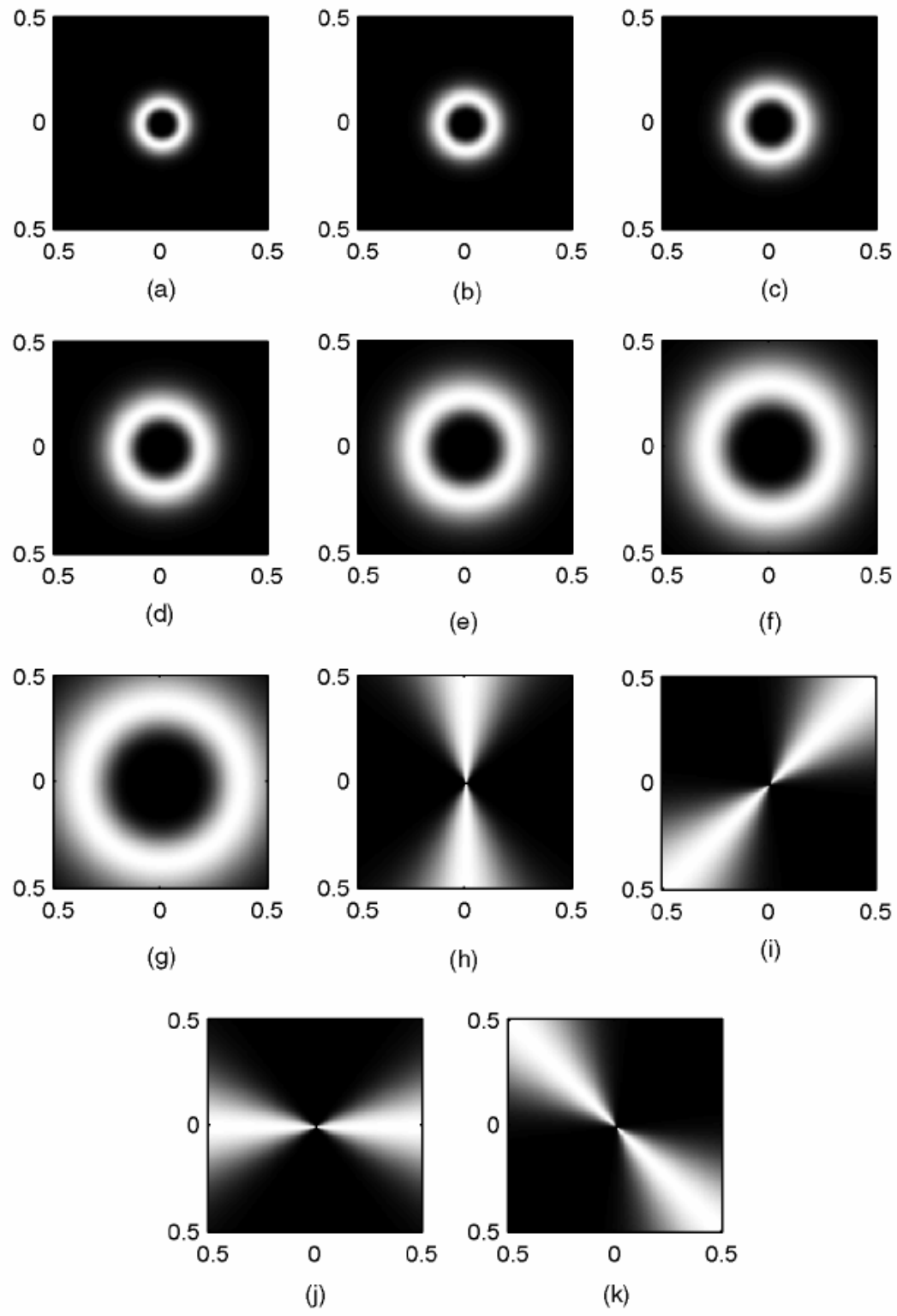


Figure 27: Ring and wedge filter bank

(from [Coggins1985]. All axes are in normalized spatial frequencies)

2.3.16 Gabor filter bank

Bovik and Jain proposed the use of the *Gabor filter bank* as a means of implementing a *texture retrieval* system [Bovik1990] [Jain1990]. This decision was based on studies on how the human visual system processes texture and how the retina was composed of cells sensitive to oriented patterns such as grating patterns [Campbell1968] [Gabor1946]. In their system, Bovik and Jain select a *Gabor filter bank* sensitive to four directions and seven frequencies, giving a total of twenty-eight filters (Figure 28). This forms a *feature vector* comprised of twenty-eight vectors for monochrome *texture data* and eighty-four vectors for colour *texture data*. While a *feature vector* constructed from a Gabor *filter bank* is not *rotation invariant* due to the directional sensitivity, it can be made to *be rotation invariant* through the use of index offseting during similarity matching. As this *texture retrieval* method conforms to two of our criteria of being both *rotation invariant* and able to work with colour *texture data*, we consider the *Gabor filter bank* to be a candidate method for *rotation invariant texture retrieval*.

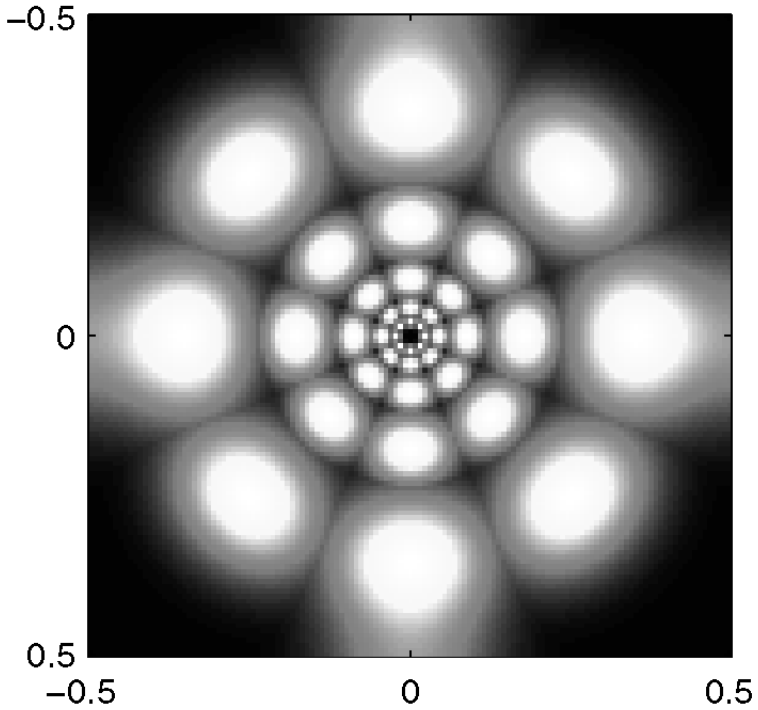


Figure 28: Dyadic bank of Gabor filters

(from [Randen1999] – All axes are in normalised spatial frequencies)

2.3.17 Shmid filter bank

Schmid introduced a *filter bank* comprised of a set of isotropic filters [Schmid2001]. This *filter bank* consists of thirteen *rotation invariant* filters, each based on a Gaussian envelope modulated by the cosine of the distance plus a constant to ensure a zero *DC component* (Figure 29). Since each filter is *rotation invariant*, the entire *filter bank* as a whole is also *rotation invariant*. For grey scale *texture images*, the *Schmid filter bank* forms a *feature vector* consisting of thirteen dimensions, For colour *texture image* data, the *Schmid filter bank* forms a *feature vector* of thirty nine dimensions is formed. As this *texture retrieval* method conforms to our criteria of being able to work with colour *texture data* and is also *rotation invariant*, we consider the *Schmid filter bank* a candidate method for *rotation invariant texture retrieval*.

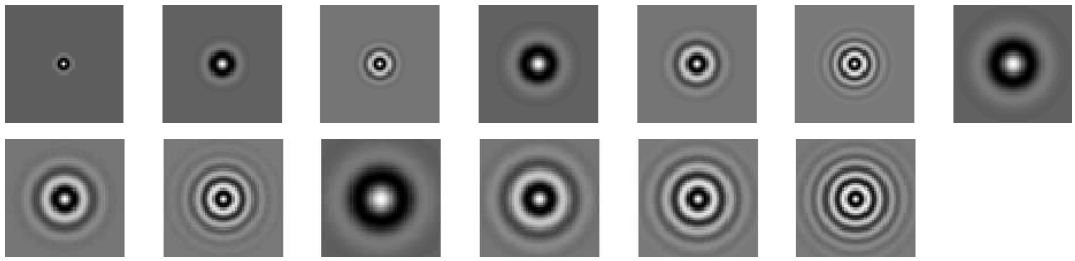


Figure 29: Schmid filter bank

(from [Varma2002])

2.3.18 Leung-Malik filter bank

Leung and Malik proposed a *filter bank* of forty-eight filters for texture filters [Leung2001]. The motivation begin the design of this *filter bank* was to construct a *filter bank* that was capable of detecting the fundamental elements of texture within a surface or *textons*. Such detail included edges, corners and spots. Each of the filters in this *filter bank* can be categorised into two types; thirty-six directional filters and twelve *rotation invariant* filters (Figure 30). The directional filters are sensitive to any one of six directions; one of three scales; and either one of two phases while the twelve *rotation invariant* filters consist of eight centre-surround derivative filters and four *Gaussian filters*. Because of the presence of the directional filters, the *Leung-Malik filter bank* is not *rotation invariant*, but through the use of index offsets during similarity matching, this obstacle can be overcome. For grey-scale *texture image* data,

the *Leung-Malik filter bank* consists of forty-eight *feature vectors*, while for colour *texture data*, the *Leung-Malik filter bank* consists of one hundred and eighty four *feature vectors*. While the *Leung-Malik filter bank* is not *rotation invariant*, it can be made to be so through the use of index offsetting during similarity searches. As the *Leung-Malik filter bank* matches our criteria of being *rotation invariant* and being able to work with colour *texture data*, we consider this method a candidate method for *rotation invariant texture retrieval*.

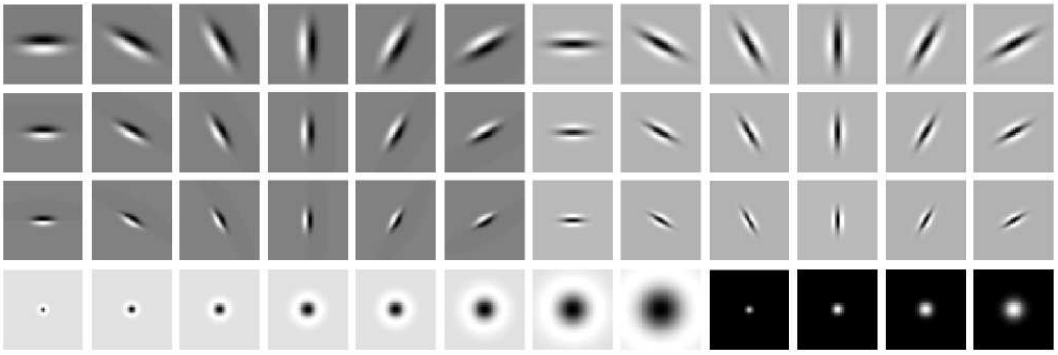


Figure 30: Leung-Malik filter bank

(from [Varma2005])

2.3.19 MR-4 and MR-8 filter banks

Varma and Zisserman proposed two modified versions of the *Leung-Malik filter bank*, the *Maximum-Response-4 filter bank* (MR-4) and *Maximum-Response-8 filter bank* (MR-8) [Varma2005]. The motivation behind the design of these two *filter banks* was to solve the problem that other *rotation invariant filter banks* were not sensitive to highly directional *texture features* such as stripes. Varma and Zisserman resolve this problem through the use of edge and bar filters. The MR-8 *filter bank* consists of the following set of filters:

- One Gaussian filter
- One Laplacian-of-Gaussian filter
- Eighteen edge filters (six directions with three frequencies)
- Eighteen bar filters (six directions with three frequencies)

The *MR-4 filter bank* is a simplified version of the *MR-8 filter bank* and consists of the following set of filters:

- One Gaussian filter
- One Laplacian-of-Gaussian filter
- Six edge filters (six directions with one frequency)
- Six bar filters (six directions with one frequency)

Both the *MR-4 filter bank* and *MR-8 filter bank* differ from other *rotation-invariant filter banks* in that they each "collapse" the responses of the edge and bar filters into a single value by only selecting the strongest response from each set of edge and bar filters with identical frequencies. Thus, for grey-scale *texture data*, the *MR-4 filter bank* forms a *feature vector* comprised of four dimensions and the MR-8 forms a *feature vector* comprised of eight dimensions. To represent colour *texture data*, the *feature vector* for the *MR-4 filter bank* requires twelve dimensions, and the *feature vector* for the *MR-8 filter bank* requires twenty-four dimensions. As both the *MR-4 filter bank* and the *MR-8 filter bank* are *rotation invariant* and can operate with colour *texture data*, we consider this a candidate method for *rotation invariant texture retrieval*.

2.3.20 The Polarogram

Davis introduced the concept of the *Polarogram* for the purposes of *texture retrieval* [Davis1981]. This method combines the transformation of a *texture image* into the *frequency domain* with the statistical analysis techniques of *histograms*. For every direction in the *frequency domain* image, an associated bin in the *Polarogram* sums the contribution of every frequency in that direction. The resulting set of data forms a graph or Polarogram (Figure 31).

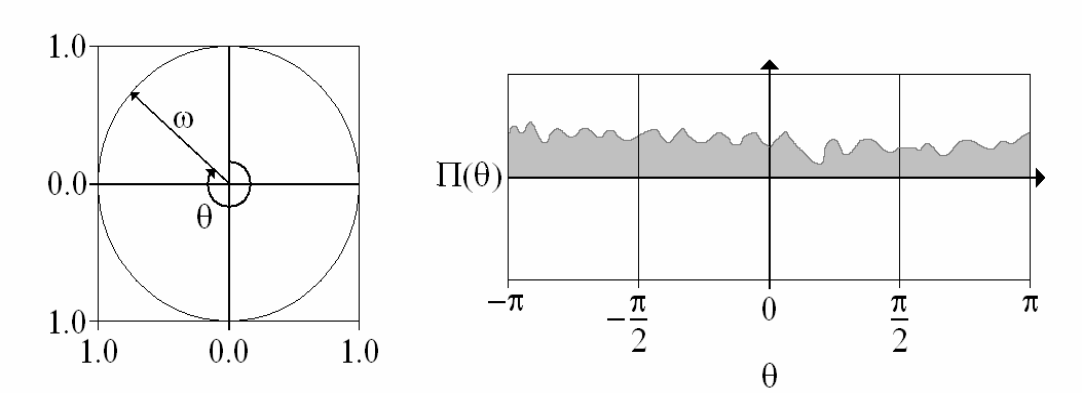


Figure 31: Calculation of the Polarogram from the frequency domain data

In his thesis, Wu chooses to focus on the application of *Polarograms* [Davis1981] for *rotation invariant texture classification* and proposes a novel surface *rotation invariant* approach to *texture classification* derived from surface derivative spectra (P and Q *gradient fields*). Wu observed that the major problem with all of the other *texture classification* methods is that none directly handle the problem of identifying either directional or isotropic textures with different *surface orientations*, and that robust *rotation invariant* features are required [Tan1995]. Wu also considers the method proposed by Smith, in which surface texture information is gained directly from *photometric stereo* and features derived from the *gradient field* (attitude, principal orientation, shape factor, and shape distribution) are used for the "quantitative analysis of repetitive surface textures" [Smith1999a].

Wu used *photometric stereo* to acquire surface *gradient field* information, Fourier analysis to transform it into the *frequency domain* and combined together using a *frequency domain* function that eliminates the directional artefacts associated with partial derivatives through the calculation of the sum of squares. Wu then uses a goodness-of-fit measure is used to compare *Polarograms* of this function are compared with those of training classes in order to provide *rotation invariant texture classification*.

2.3.21 Summary

In the previous sections we have identified nine candidate methods for *rotation invariant texture retrieval* and discussed their ability to encode data concerning periodicity and colour while being *rotation invariant*. We list all nine methods in (Table 4) to summarize these findings.

Method	**Frequency domain Implementation**	**All filters Rotation-invariant**	**All filters can detect periodic Patterns**
Colour Histograms	No	Yes	No
Ring Filter bank	Yes	Yes	Yes
Wedge Filter bank	Yes	No	No
Gabor Filter bank	Yes	No	Yes
Schmid Filter bank	Yes	Yes	Yes
Leung-Malik filter bank	Yes	No	Yes
MR-4 filter bank	Yes	No	Yes
MR-8 filter bank	Yes	No	Yes
Polarogram	Yes	Yes	No
Combined	Both	No	No

Table 4: List of candidate methods for texture retrieval

It should be noted that any linear convolution filter in the *spatial domain* may also be performed in the *frequency domain*. However, for small filters (3x3 sample points) it is more efficient to apply the filter in the *spatial domain* than it is in the *frequency domain*.

Colour histograms operate in the *spatial domain* and are naturally *rotation invariant* as they operate on pixel intensity data alone. For this reason, we consider this method suitable for our needs.

With the exception of the *histogram* all of the above techniques are based on linear filters and are such they merely divide the power spectra up into different “chunks” within which they calculate the variance of the signal that is left after application of the relevant band pass filter. Ring type filters (including the *Polarogram*) are by their nature rotational insensitive and therefore not *rotation invariant*. *Wedge filters* and other rotational sensitive filters require some offsetting technique in order to make them

rotation invariant. It is therefore just a question of which filter sets are better at discriminating between the different power spectra characteristics of textures that will determine which are most suitable for our task. However, it should be noted that these filters are not normally applied to *micro-geometry* data and as such we need to apply the techniques of Wu in order to remove the directional bias present in the *bumpmap* data.

Having reviewed all the literature related to *texture retrieval*, we now proceed to discuss how the current information retrieval system evaluation methods can be used to identify the most suitable *texture retrieval* methods for virtual textile database queries.

2.4 Review of current information retrieval system implementations

As the ability to capture and store feature-rich data such as images, video, audio, multi-channel sensors and three-dimensional geometry has increased due to the availability of high capacity storage and acquisition equipment, the need for users to be able to index and search this data has also increased. Unlike traditional alphabetic and numeric value based databases where a search can be implemented using a combination of basic arithmetic comparison operators, searching feature-rich data requires complex similarity operators customised to each type of data. One solution to this problem is to construct a modular information retrieval system which separates the generic database management tasks from the custom feature extraction and similarity operator functions required for different types of multimedia data such as images, video and audio. This enables researchers to focus their attention on the development of these functions rather than on the entire information retrieval system. In this thesis, we take advantage of this design and keep the scope of this thesis purely to the identification of the *feature vectors* most suited to being extracted from textile *image data* and for the implementation of *rotation invariant* similarity operators. Thus there are two issues which must be resolved in order to implement such a system; which *filter banks* and frequencies should be used and how important is colour information when attempting *rotation invariant* texture retrieval. However, before we investigate their performance in detail in Chapter 7, we will briefly examine the Ferret toolkit and review the techniques that researchers normally employ to evaluate retrieval systems.

2.4.1 Ferret: A toolkit for content-based similarity searches

The Ferret Toolkit has been designed to allow system designers to rapidly construct search engines with such feature-rich data [Lv2006]. The Ferret toolkit separates the task of implementing an information retrieval into several layers (Figure 32).

At the top layer are the data acquisition, web interface and performance evaluation tools. The data acquisition module allows the database system to receive new data and pass it through to the similarity based search engine. The web interface allows the client side of the database to be implemented using standard web page layout. The performance evaluation toolkit allows maintainers to generate batch queries to compare performance and search quality results against benchmark values. This is of particular interest to this thesis due to the need to evaluate the performance of the candidate *texture retrieval* methods.

Below the web interface and performance evaluation tool is the command line query interface which communicates with the similarity search API. Beneath the similarity search API lie the generic functionality of the data base system; the attribute based search tool, metadata management (transaction management) and the core similarity search engine. The core similarity engine is responsible for the construction of 'sketches' from *feature vector* data and implements the generic similarity search functionality, which enables the search engine to work with file objects. Ferret optimises the calculation of similarity comparisons by generating and comparing 'sketches' of each *feature vectors* before comparing the actual *feature vectors*. Each 'sketch' consists of a bit vector of a specific length derived from the associated *feature vector*. This allows a similarity comparison to be performed between two 'sketches' by using the Hamming distance calculation.

Along with the performance evaluation tool, the two plug-in components are of particular interest to this thesis; the first is the segmentation and feature extraction tools, while the second is the distance functions used to perform similarity matching. In this thesis, our candidate *texture retrieval* functions correspond to these two components. However, we choose not to use this implementation of a complete information retrieval system as the scope of this thesis is to identify *feature vectors* suitable for the

implementation of *rotation invariant texture retrieval* of textile data and that to do so would have required considerable more resource than was available.

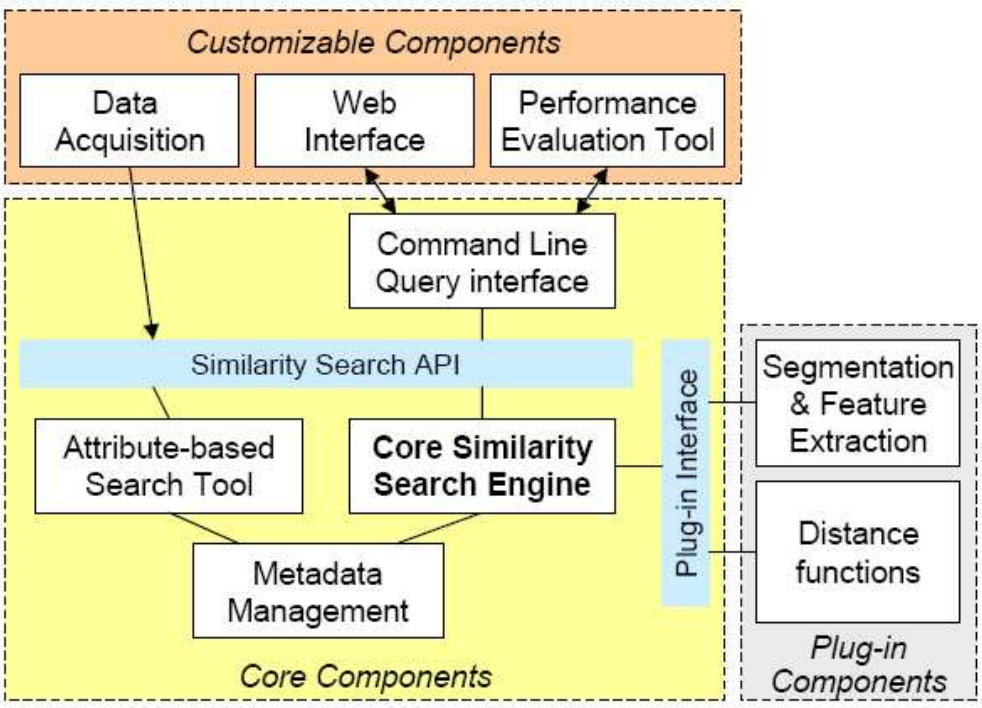


Figure 32: Architecture of the Ferret Toolkit for Content-Based Similarity Searches

2.4.2 Measuring the accuracy rate of information retrieval systems

In the previous section, we described how modern information retrieval systems are implemented through the separation of the system into separate modules, with the Ferret toolkit as a practical example. In this thesis, our candidate *texture retrieval* methods correspond to both the segmentation and feature extraction modules and the distance function modules of this toolkit.

However, in order to determine which of the candidate *texture retrieval* are most suited to this task, we must have some measure of being able to benchmark or measure statistically the success rate of each *texture retrieval* method. Buckland describes a method of achieving this through the concepts of *recall* and *precision* when applied to information retrieval system queries [Buckland1993]. This has now become the standard way of comparing the performance of different information retrieval systems

for a wide variety of content including text, images and video. One such working group, Text Retrieval Conference (TREC) conducts an annual survey of the performance of video retrieval systems [Smeaton2004] [Smeaton2006]. During each survey, sample datasets and queries are distributed out to researchers, who then return the results of their implementations through the measurement of *precision*, *recall* and the *F-measure*.

We describe the method of calculating the *precision*, *recall*, *accuracy* and *fallout* for a selected information retrieval method. For any particular search query, four results are generated:

- The number of items relevant and retrieved (true-positive)
- The number of items not relevant and retrieved (false-positive)
- The number of items relevant and not retrieved (false-negative)
- The number of item relevant and not retrieved (true-negative)

Of particular concern to the designer of an information retrieval system, is the rate of failure, when either a relevant item is not retrieved (false-negative), or when a non-relevant item is retrieved (false-positive). False-positive outcomes are referred to as Type I Errors, while false-negative outcomes as Type II Errors. We can place these results into a table and assign them labels (Table 5).

		Expected result / classification	
		Relevant	**Not Relevant**
Obtained result / classification	**Relevant**	**tp** (true positive)	**fp** (false positive) Type I Error
	Not Relevant	**fn** (false negative) Type II Error	**tn** (true negative)

Table 5: Classification table for information retrieval

From these four values, four statistical values can be determined; *precision*, *recall*, *accuracy* and *fallout*. Each of these values is a fraction in the range 0.0 to 1.0. *Precision* is the ability of the retrieval system to return only items relevant to the search query. *Recall* is the ability of the retrieval system to retrieve every relevant item to the search query. *Fall-out* is the ability of the retrieval system to retrieve non-relevant items to the

current search query. *Accuracy* is the ability of the retrieval system to retrieve relevant items and discard non-relevent items.

$$\text{Precision} = \frac{tp}{tp + fp} \qquad (2.4.2.1)$$

$$\text{Recall} = \frac{tp}{tp + fn} \qquad (2.4.2.2)$$

$$\text{Fall-out} = \frac{fp}{fp + tn} \qquad (2.4.2.3)$$

$$\text{Accuracy} = \frac{tp + tn}{tp + fp + fn + tn} \qquad (2.4.2.4)$$

Where: tp is the number of true positive results,

fp is the number of false positive results,

tn is the number of true negative results,

and fn is the number of false negative results.

Measurement of the *accuracy rate* of a *texture retrieval* method is performed by generating a large set of test queries and comparing the expected results returned against the actual returned results. These results may be averaged and used to create two types of graph:

- Receiver-operator-Characteristic graphs (ROC graphs)
- Recall-Precision graphs

With ROC graphs, the Y-axis is in the range 0.0 to 1.0 and represents the true-positive rate, while the X-axis in in the range 0.0 to 1.0 and represents the false-positive rate. A typical graph curves will start at the coordinate (0.0, 0.0) and finish at the coordinate (1.0, 1.0). Information retrieval methods with a high *accuracy rate* will have a curve that rapidly approaches the top left corner of the graph (1.0, 0.0) before rapidly leveling out towards the top right of the graph (1.0, 1.0). Different retrieval methods may be compared against each other by comparing the position and gradient of such graph curves.

Recall-Precision graphs use the Y-axis to represent the *precision* and the X-axis to represent the *recall*. A typical *recall-precision* graph will have a graph curve that begins somewhere near the top left of the graph (0.0,1.0) and moves towards (1.0,0.0). As with the ROC graph, it is possible to compare retrieval methods by comparing the slope and gradient of such graph curves.

2.4.3 A survey of texture retrieval systems

While there is a vast amount of literature related to *texture retrieval* using standard images illuminated by natural or artificial light (the *albedo*), less research has been conducted on *rotation invariant texture* retrieval, especially with *3D surface representations.*

Dana investigated applications of the BTF to represent and relight 3D surfaces [Dana1997] [Dana1999] and also the classification of 3D texture using *histograms* [Dana1998]. As this system was designed for image samples with constant *albedo*, this system used only a grey-scale *histogram* and to eliminate the effect of *self-shadowing*, pixel values below a selected threshold are discarded. This method has the advantage of not requiring *image data* to be transformed into the *frequency domain*, but has the disadvantage of not working with images with varying *albedo*.

McGunnigle investigated the use of *photometric-stereo* techniques combined with the use of a *feature vector* comprised of thirty-two *Gabor filters* to implement *texture classification* with illuminant direction invariance in the *spatial domain* [McGunnigle1997] [McGunnigle1998] [McGunnigle1999]. This decision was based on the research documented by Jain and Farrokhia [Jain1991] which aimed to develop a *texture retrieval* system based upon biological models. This method has the advantage of modeling human perception of textures but at the disadvantage of requiring a large *feature vector*.

Varma *investigated texture classification* of images from the Columbia-Utrecht database using *texton dictionaries* generated from k-means clustering techniques

[Varma2002a] [Varma2005]. In the more recent paper, Varma performs a comparison of the *Leung-Malik filter bank*, the *Schmid filter bank*, the *MR4 filter bank* and the *MR8 filter bank* in order to determine the *texture classification* rates of each method. In each case, *texture classification* operates in two stages. In the first stage, the training stage, the *texton dictionary* is generated from the set of existing images. In the second stage, the retrieval stage, textures are classified using the *texton dictionary*. This system has the advantage of using *rotation invariant texture retrieval* but at the disadvantage of requiring a learning stage.

Wu investigated the use of *photometric stereo* to implement *rotation-invariant* classification of 3D surface texture using the polar spectrum (the *Polarogram*) and the radial spectrum in conjunction with both gradient and *albedo* data [Wu2003]. In this system, the two dimensional image generated from transforming textures images into the *frequency domain* is reduced in complexity from two dimensions to one dimension. This method has the advantage of avoiding the computation time required to compare a single *texture image* against an entire *filter bank*. It also has the advantage of being made *rotation invariant* through the use of index offseting.

Drbohlav investigated the use of single training images to implement illumination invariant *texture classification* [Drbohlav2005] [Drbohlav2005b]. Using this method, the class that every new image added to the database is determined by calculating the *feature vector* distance between the image and each of the pre-existing training images. The training image with the shortest *feature vector* distance identifies the class that the new image belongs to. The *feature vector* used by this system is comprised from fifteen *Gabor filters* arranged as three rings of five filters each separated by 36 degrees. This method has the advantage of using a smaller *feature vector* than the system described by McGunnigle, but with the disadvantage of requiring a training set of images.

We extend the research carried out by these papers by investigating the application of *rotation invariant texture retrieval* using p and q *gradient fields* in both the *frequency domain* and the *spatial domain*. As our criteria require the use of colour *texture data*, we choose not to use the grey-scale *histogram* method described by Dana, but instead use the *colour histogram* method described by Funt and Swain. We choose to use a large

Gabor filter bank as described by McGunnigle rather than the smaller *filter bank* as described by Drbohlav. Based upon the research conducted by Varma, we choose to investigate the *Schmid filter bank*, the *Leung-Malik filter bank*, the *MR-8 filter bank* and the *MR-4 filter bank* but without the use of a set of training images. Finally, based upon the research conducted by Wu, we choose to investigate the use of the *Polarogram* as a means of reducing the size (or dimensionality) of the *feature vector* for *texture retrieval*. As we were unable to find any documented research in the combined use of colour (*albedo*) and *micro-geometry* (*gradient field* data, *normalmaps* and *heightmaps*) we choose to investigate the use of *rotation invariant texture retrieval* methods with this data. We also attempt to combine all ten methods together in an attempt to improve performance with the disadvantage of increasing the size of the *feature vector*.

2.4.4 Summary

In this section, we have performed a review of current information retrieval systems designed to work with generic types of data. We have identified that both the feature extraction and segmentation module and the distance function module of the Ferret toolkit correspond to the *feature vectors* required for *rotation invariant texture retrieval*. We have also identified the need to measure the *accuracy rates* of each of the selected *texture retrieval* methods based upon the standard measures of *precision* and *recall* in order to compare and evaluate the performance of the candidate *texture retrieval* methods. It is our intention to combine these together in order to evaluate and compare the most suitable *rotation invariant texture retrieval* method for virtual textile databases. We will present out findings using both ROC and Precision-Recall graphs.

Having performed a review of information retrieval systems and benchmarking, we now proceed to present the conclusions from our literature survey.

2.5 Conclusions

In this chapter, we have performed a literature review in the main research fields relevant to our thesis (*3D surface representations*, textile visualisation, *texture retrieval* and *information retrieval database validation methods*).

Our summary conclusions are as follows:

2.5.1 3D Surface Representations

We choose to use *relief mapping* to represent textile samples as this is the only *3D surface representation* that can be obtained using a simple camera setup and that represents both the *albedo and micro-geometry* of textile samples and thus allow the implementation of *self-shadowing* and *self-occlusion*. As *relief-mapping* is an extension of both *texture mapping* and *bump-mapping*, we also choose to investigate both of these simple methods. However, one problem with the use of *normalmap* data for *rotation invariant texture retrieval* is that there is a directional dependency due to the use of axis coordinates to represent the per-pixel *outward normals*. We describe this process in Chapter 3, and discuss suitable methods for acquiring these data in Chapters 4 and 5.

2.5.2 3D Visualisation

For the *real-time* visualisation of the *3D surface representations*, we choose to use the generic texture description provided by Stürzlinger. We base this decision due to the ability of the texture description to represent every possible combination of surface appearance from matte surfaces to shiny reflective and translucent surfaces. To improve the visual realism of the visualisation system, we also choose to implement hard shadows using the *shadow mapping* method. We base this decision due to the ability of the *shadow-mapping* method to support dynamic light sources and not require any pre-calculation of scene geometry along with being supported by 3D graphics hardware. In order to combine together the rendering of *relief-mapping* and *hard shadows* with multiple light sources will require the development of a rendering algorithms and shaders. While *shadow-mapping* allows *3D geometric objects* to project shadows onto each other and concave *3D geometric objects* to project shadows onto itself, it does not solve the problem of self-shadowing of the *micro-geometry*. This can only be performed

through the use of *relief-mapping*. The requirement for *relief-mapping* and *shadow-mapping* techniques thus necessitates the use of a *programmable graphics accelerator board* in order to implement per-pixel lighting. Combining these two techniques is an issue that remains to be resolved.

Furthermore, as the visual properties of textiles as encoded by the *micro-geometry* are often more obvious when presented on a curved surface we will investigate the use of *3D geometric objects* represented using Bézier patches as the use of *parametric surfaces* with *control points* will allow for the rapid construction and simple animation of complex geometry. To generate the *tangent space* required for *both relief-mapping* and *bump-mapping* will require the development of *parametric surface* evaluation algorithms. We describe our approach to integrating *micro-geometry* and *macro-geometry* in Chapter 5, then present the integrated visualisation system in Chapter 6.

2.5.3 Texture retrieval system

We have decided not to implement a complete information retrieval system with a complete front-end user interface due to the time constraints imposed by this thesis, and so we choose not make use of the Ferret toolkit. Instead we choose to focus our attention on the implementation of feature extraction and similarity operators in order to determine which *filter bank* is best for the analysis of texture periodicity. To this aim we will investigate the use of *rotation invariant filter banks* in the *frequency domain,* and *colour histograms* in the *spatial domain* as we consider periodicity and colour important to texture retrieval. These methods will include the *ring filter bank,* the *wedge filter bank*, the *Gabor filter bank*, the *Schmid filter bank*, the *Leung-Malik filter bank*, the *MR4 filter bank*, the *MR8 filter bank* and the *Polarogram*. To benchmark the operation of each of these *texture retrieval* methods, we will make use of the *precision* and *recall* measurement techniques as described earlier. We describe our research in this area in Chapter 7 of this thesis.

Chapter 3 – Data Requirements

3.1 Introduction

In the previous chapter we reviewed several candidate representations of virtual textiles that would be suitable for both *texture retrieval* and rendering purposes, and identified that *reliefmaps*, *bumpmaps* and *texturemaps* would all warrant further investigation. The purpose of this chapter is to review possible sources for thesis data both in terms of existing public databases and acquisition methods, but before this, we will quickly discuss the requirements we have of these data. From our literature survey, we have identified the following requirements:

The data must be able to encode relief-maps, bump-maps, and colour texture maps in a form suitable for use with current generation *programmable graphics accelerator boards* and for the derivation of suitable *feature vectors* and at sufficient resolution. For visualisation, we require height data (h), unit surface normals ($\mathbf{n}$), and colour *albedo* (r,g,b) sampled on a regular Cartesian grid in x and y.

The *feature vector* will be used to encode periodic characteristics of weaves and other fabrics together with colour data in a rotation invariant manner. As many features are computed in the *frequency domain*, it will be more efficient if the data are organized in a form suitable for processing by FFT ie. a regular grid of side 2^n. It should be noted that the x and y components of the *surface normal* (n_x, n_y) are naturally directional and are therefore not directly suited to computing rotationally invariant features. This will be addressed in Chapter 7.

The above requirements mean that we require height, normal and colour data sampled in a regular grid at a resolution of 512 x 512 or greater.

3.2 Existing databases

While many *texture databases* already exist; the Brodatz collection [Brodatz1966], MeasTex [Ohanian1992] [Smith1997], the CUReT data set [Dana1997] [Dana1999], the VisTex Database [Vistex2002], the Outex data set [Ojala1996] [Ojala2002] [Cola2004], PhoTex [McGunnigle2001], PMTex [Wu2003] very few if any deal exclusively with textiles. Instead, many consist of a wide variety of colour image samples with objects ranging from vegetation and foodstuffs to natural and human architecture (VisTex, CURet), while others consist entirely of monochromatic *image data* (Brodatz, MeasTex, PhoTex, PMTex, OUTex) and/or only have one orientation of each object. Furthermore, these databases do not contain explicit gradient data. In a few cases such as CURET, data from multiple sample orientations is available and in principle as normals could have been estimated from these images but this requires registration and interpretation of what is already comparatively low resolution data.

As the focus of this thesis is exclusively on virtual textile databases and in particular the use of micro-geometry data, we consider none of the existing *texture databases* to be suitable for the research described in this thesis.

3.3 Acquisition techniques

The main requirements for an acquisition system are that it should produce the data as described in the introduction and that it should be economic and simple to set up. At present, there are a wide variety of methods for the acquisition of the micro-geometry of the surfaces of solid objects. These include tunneling electron microscopes, laser and ultrasound scanners. However, all of these methods have the disadvantage of requiring expensive data acquisition systems, and in the case of tunneling electron microscopes, require that each sample be coated with a thin layer of gold before being placed in a vacuum chamber. With some laser scanners, there is also the disadvantage that the

system will not generate a regular mesh of data points. Other systems such as hand-held digitizer pens have the disadvantage of requiring manual sampling with each single sample point taking five seconds to acquire. As our requirements are that at least 256,000 sample points must be acquired, this method is not practical. The only alternative to all of these methods are photographic techniques such as *photometric stereo*. This method provides surface normal and colour *albedo* data and requires only a digital camera as an acquisition source, along with three digital images of the target surface taken with different light source angles. Note that the height data may be obtained through integration [Frankot1988].

Thus when compared to the architecture of the Ferret toolkit, we observe that the Data Acquisition module of Ferret matches the *Photometric Stereo* stage of our thesis, while the Feature Extraction modules also match. However, our framework differs from Ferret in that we use *photometric stereo* to acquire *albedo, gradient field* and *heightmap* information, and 3D rendering techniques to visualize this *texture data* applied onto *3D geometric models*.

For each texture sample, we acquire four images of the texture at different orientations using a fixed camera (Canon SLR camera). For each orientation of the textile sample, we capture a set of three images under different lighting orientations. We refer to these *image data* sets as "*photometric images"* [Woodham1980]. These are sufficient to accurately acquire the *albedo*, *gradient field* and *heightmap* data. We provide detailed explanation of the process used to convert these *photometric images* into *albedo*, *gradient field* and *heightmap* data in (Chapter 4). We acquire all images at an original pixel resolution of 1280x1024 at 24-bits per pixel, as this is the optimum resolution to capture the repetitive nature of a regular pattern as seen from the camera while maintaining the highest level of detail of *micro-geometry*. We represent all samples in the database as true colour quality images at a pixel resolution of 512x512 as a square dimensions is required for our discrete FFT conversion process. Our method of acquiring and generating texture samples is based upon the conclusions reached by the research conducted by the other researchers at Texturelab; Gullón, McGunnigle and Spence investigated modifications to the Frankot integration method [Gullón2003] [McGunnigle1998] [Spence2005]. McGunnigle initially set up the camera system and created the Photex database [McGunnigle2001]. Spence developed the software to

convert *photometric stereo image data* sets into *gradient field* and *heightmap* data [Spence2005].

To determine whether an *image data* set is added to the Virtex *texture database* or not, we use the following criteria. The texture sample must consist of a regular pattern and not a simple image. Thus, knitted cartoon images are not suitable for inclusion in our database. Secondly, the texture sample must fill the entire area of the texture frame with a solid pattern. Thus, textile samples which are comprised of a fishnet pattern that can be seen through or which are too small to fill the frame of the digital camera are not suitable for inclusion in our database. We choose a large number of textile samples to provide a wide variety of texture patterns. These include fine-weave patterns with a near Lambertian *reflectance model*; rough weave patterns that create *self-shadowing* and *inter-reflectance* lighting effects; textiles with semi-transparent threads that allow for *translucency*, and others with a glossy coating that provide specular highlights. Altogether, we acquire twenty textile samples with each sample taken at four randomly chosen orientations. As each textile sample requires three *photometric images*, this results in a total of 240 images. This relatively low number of samples is due to the time taken to process each textile sample, as described in Chapter 4 – Image Acquisition using *Photometric Stereo*.

We present the complete set of textile images in (Appendix B – The Texture Dataset)

3.4 Conclusion

In this chapter, we defined our data requirements for the acquisition, retrieval and *3D visualisation* of textile samples. Based upon these requirements, we introduced the data environment used to perform all *texture retrieval* experiments in this thesis.

The data representation enables integration of both *texture retrieval* methods based upon 3D surface texture and *3D visualisation* methods in order to implement *real-time 3D visualisation* of virtual textile databases. The data acquisition and processing thus consists of three stages: the first stage acquires a set of *3D surface representations* using

photometric stereo, the second stage implements a *texture retrieval* system obtained using the images acquired by the first stage, and the third stage consists of the visualisation of these texture on *3D geometric models*. However, the use of *normalmaps* introduces the problem of directional artifacts, which must be resolved if *rotation invariant texture retrieval* is to be implemented successfully.

We now proceed to describe how *photometric stereo* techniques may be used to acquire the data required to represent the *3D surface representation* of textile samples. This information includes the *albedo*, *surface normals* represented as a *normalmap* and also a *heightmap*.

Chapter 4 – Data Acquisition using Photometric Stereo

4.1 Image Acquisition using Photometric Stereo

In Chapter 3, we described the generic data representation for this thesis that would allow us to use a single *3D surface representation* in conjunction with *texture retrieval* and *3D visualisation* techniques. In this chapter, we describe how a standard digital camera and light source can be used with *photometric stereo* to acquire the *albedo*, *heightmap* and *normalmap* components of the *3D surface representation* and how these data may be converted for use with *current programmable graphics accelerator boards*. This forms the first stage of our data representation (Figure 33).

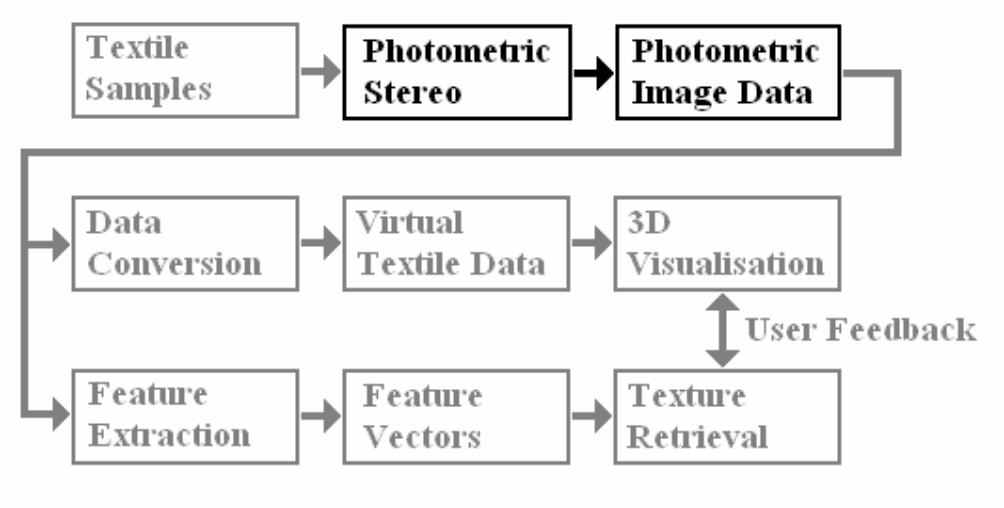


Figure 33: The image acquisition stage of our data representation

One method of capturing the *micro-geometry* of a texture is through the technique known as *photometric stereo*. This method requires three pieces of equipment: (1) a camera that is fixed directly above the textile sample, (2) the textile sample itself fixed so that it does not move between image captures, and (3) a light source that is a constant distance away from the textile sample, but is free to rotate in a circular path (Figure 34).

Using *photometric stereo*, three images are captured, each with the light source at the same distance and height, but with a different direction. By using *photometric stereo* methods [Woodham1980] [McGunnigle1997] [McGunnigle1999] [Spence2005] [Wu2003], it is possible to estimate the p and q *gradient fields* (or partial derivatives) of each pixel in the image. From these *gradient fields*, it is possible to calculate a *normalmap*. Using the *normalmap*, and *integration methods*, it is possible to calculate the *albedo* and *heightmap* of the image. Woodham demonstrated that it was possible to recover both the *p gradient field* and *q gradient field* information, as well as the *albedo* at every image point from just three images [Woodham1980] illuminated by three different non-coplanar light sources.

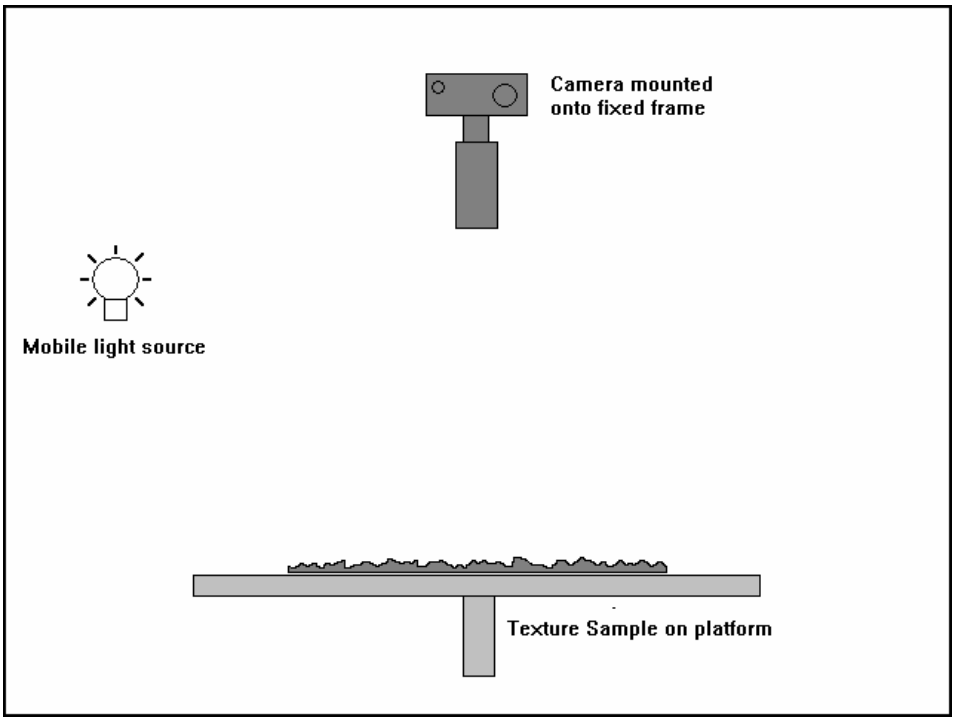


Figure 34: The photometric stereo apparatus

(Due to the preliminary nature of this work, only a figure diagram is available)

We can define the *illumination vector* in terms of tilt and slant angles:

$$\mathbf{l} = \begin{bmatrix} l_x \\ l_y \\ l_z \end{bmatrix} = \begin{bmatrix} \cos\tau \sin\sigma \\ \sin\tau \sin\sigma \\ \cos\sigma \end{bmatrix} \qquad (4.1.1)$$

Where: τ is the tilt angle,

σ is the slant angle,

$\mathbf{l}$ is the *illumination vector*,

and (l_x, l_y, l_z) are the individual elements of the *illumination vector*

The three directions of incident illumination $\mathbf{l_1}, \mathbf{l_2}$ and $\mathbf{l_3}$ are defined as:

$$\begin{aligned} \mathbf{l_1} &= [l_{x1} \quad l_{y1} \quad l_{z1}] \\ \mathbf{l}_2 &= [l_{x2} \quad l_{y2} \quad l_{z2}] \\ \mathbf{l}_3 &= [l_{x3} \quad l_{y3} \quad l_{z3}] \end{aligned} \qquad (4.1.2)$$

Combined together, these form the light matrix L:

$$L = \begin{bmatrix} \mathbf{l_1} \\ \mathbf{l_2} \\ \mathbf{l_3} \end{bmatrix} = \begin{bmatrix} l_{x1} & l_{y1} & l_{z1} \\ l_{x2} & l_{y2} & l_{z2} \\ l_{x3} & l_{y3} & l_{z3} \end{bmatrix} \qquad (4.1.3)$$

We define the column vector $\mathbf{n}$ as the unit *surface normal* at (x, y) as:

$$\mathbf{n} = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \qquad (4.1.4)$$

Assuming Lambertian reflectance, the intensity of the three images at the point (x, y) are related to the *albedo* k_p by the equation:

$$I = \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} = k_\rho . L . \mathbf{n} \qquad (4.1.5)$$

However, the light matrix L is known through the positioning of each light source, and the intensities I are known through the acquisition of *photometric images*. This leaves the both the *albedo* k_ρ and *surface normal* $\mathbf{n}$ as unknown.

Rearranging this equation gives the "scaled surface normal":

$$\mathbf{t} = k_\rho . \mathbf{n} = L^{-1} . I \qquad (4.1.6)$$

Providing that the three vectors $\mathbf{l_1}$, $\mathbf{l_2}$ and $\mathbf{l_3}$ are not collinear, then the inverse L^{-1} exists, and this equation is solvable. Then the *albedo* k_ρ is determined from:

$$k_\rho = |\mathbf{t}| = \sqrt{t_x^2 + t_y^2 + t_z^2} \qquad (4.1.7)$$

Where (t_x, t_y, t_z) are the components of the scaled *surface normal* $\mathbf{t}$

The unit surface normal $\mathbf{n}$ can be simply derived:

$$\mathbf{n} = \frac{\mathbf{t}}{|\mathbf{t}|} \qquad (4.1.8)$$

Integration of the *surface normal* using Fourier analysis [Frankot1988] [Gullón2008] allows the surface *heightmap* to be calculated from the p and q *gradient fields*, which are defined as:

$$p = \frac{\partial x}{\partial z} \qquad \text{and} \qquad q = \frac{\partial y}{\partial z} \qquad (4.1.9)$$

and may be simply derived from the surface normal ($\mathbf{n}$).

For this thesis, we chose to use a command line utility program written in the 'C' programming language running on Linux [Spence2005]. Input to this utility program consists of the *three photometric images* stored in 16-bit format raw image camera files, the *spherical coordinates* of each light source expressed as floating-point values and the output consists of an *albedo* image stored as a 32-bit floating point image, two *gradient field images* in p and q stored as 32-bit floating point images and the *heightmap*.

4.2 Using photometric stereo data with 3D graphics accelerator boards

We begin this section by describing the decision made for the representation and processing of *texture data* required for the *relief-mapping* of textile samples. We have chosen to use these techniques in this thesis due to the fact that the standard methods of *texture-mapping* do not allow for the accurate photorealistic visualisation of the *micro-geometry* of textile samples. This is particularly important when curved geometry is rendered using point and directional light sources. Due to their intrinsic method of manufacture, textiles have a complex surface appearance which can only be captured and reproduced through the use of three types of high-resolution image, which we refer to as *albedo*, *bump-map* and *height-map* images. The *albedo image* defines the appearance of the *surface material* under optimal ambient lighting conditions. The *bump-map* image defines the individual *surface normal* for each sample point in the *albedo* image. The *height-map* defines the height of each individual sample in the *albedo* image relative to a fixed reference point. Rendering 3D geometry in *real-time* with such *surface materials* requires the use of a *programmable graphics accelerator board.* Implementing either Blinn *bump-mapping* or *relief-mapping* on a *programmable graphics accelerator board* requires a minimum of two images. For Blinn *bump-mapping*, only the *albedo* and *normal-map* are required, while for *relief-mapping*, the *height-map* is also required along with the *albedo* and *normal-map*. We present example *albedo*, *bump-map* and *height-map* images in (Figure 38) and (Figure 39).

In the previous section, we described how to use the *photometric stereo* process to convert three intensity images into the *albedo* and *gradient field* represented as 32-bit floating point data. However, in order to convert these images for use with *programmable graphics accelerator boards*, the *albedo* image must be in an 8-bit RGB or RGBA image and the *normalmap* must be combined together with the *heightmap* to form a single 8-bit per channel RGBA image in order for use with hardware accelerated *bump-mapping* or *relief-mapping*

In order to make both the *albedo* image suitable for use with a *programmable graphics accelerator board*, we convert the *albedo* into an 8-bit RGBA texture. Conversion from floating point to 8-bit data simply involves the normalization of the floating point data

to the range 0.0 to 1.0, rescaling to the range 0 to 255, and then conversion to 8-bit integer data.

We use the *alpha channel* of the *albedo* texture to represent the *gloss map* (also known as the *specularity map).* If no *gloss map* is present, this field is assumed to be 1.0 as attempts to read the *alpha channel* of a RGB texture will always return the constant 1.0.

For both *bump-mapping* and *relief-mapping* methods, the second texture is referred to as the *normalmap* and encodes the *surface normal* of each pixel as either a floating point or eight-bit signed vector in the RGB channel. Because each *surface normal* is a unit vector, each component will range from –1.0 to 1.0.

The resulting unit vector is then scaled and biased for compression into an 8-bit signed RGB colour value. The *surface normal* is encoded be converted each component into an 8-bit unsigned value by adding 1.0 and multiplying by 127. For *relief-mapping*, the *surface normal* is represented the same as before, but the *alpha channel* is now used to represent the *heightmap*, encoded into the range 0.0 to 1.0.

We present a diagram of the complete conversion process in (Figure 35). We present a typical textile sample that has been framed and fixed to a backing board in (Figure 36). Using the digital camera and multiple light sources, we acquire a set of four *photometric images* (Figure 37) of which only three are used to generate the **p** and **q** *gradient field* images (Figure 38). From these two *gradient field* images, the *albedo*, *bumpmap* and *heightmap* images are generated.

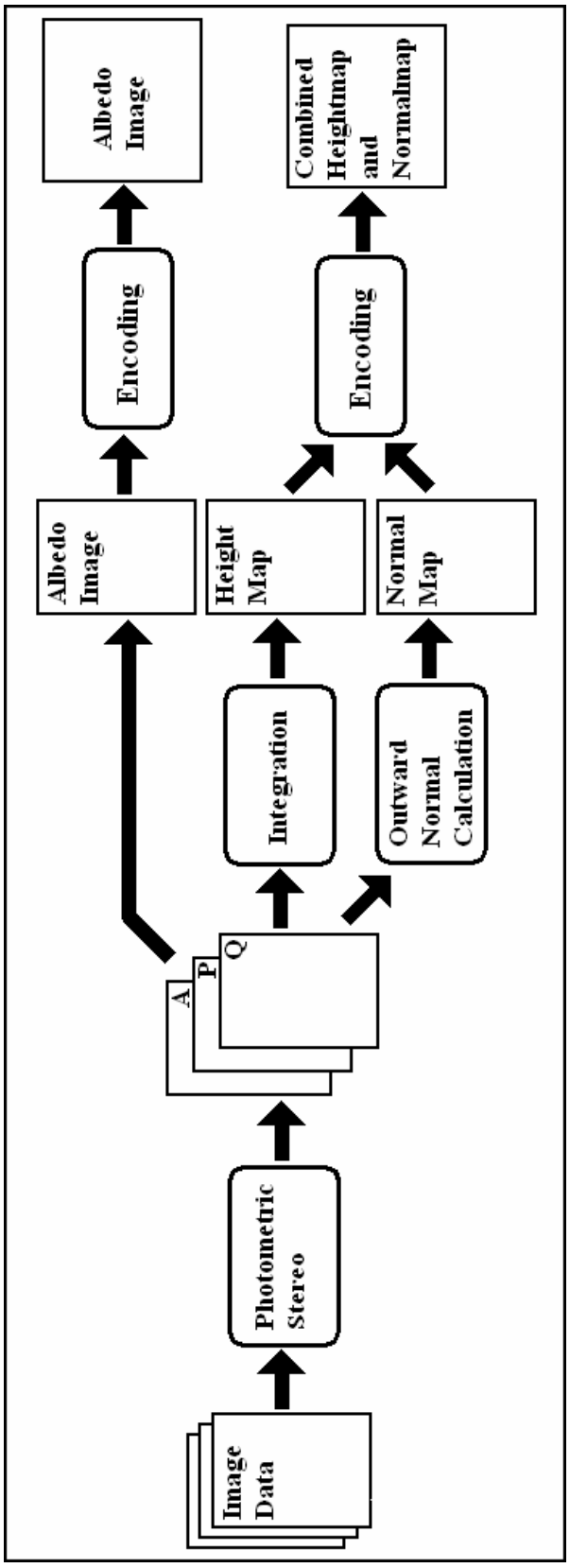


Figure 35: Photometric Stereo Capture Pipeline

Figure 36: Textile sample used for photometric stereo

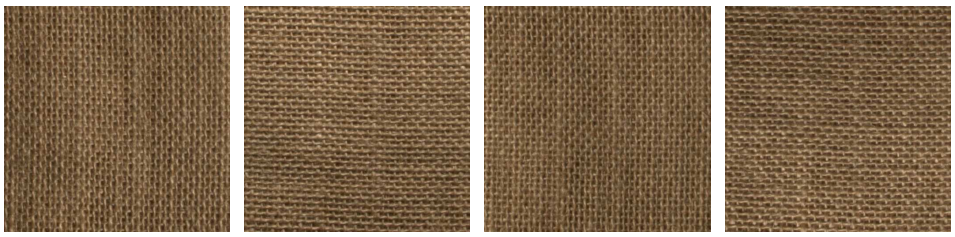


Figure 37: The set of photometric images each with a different light source direction

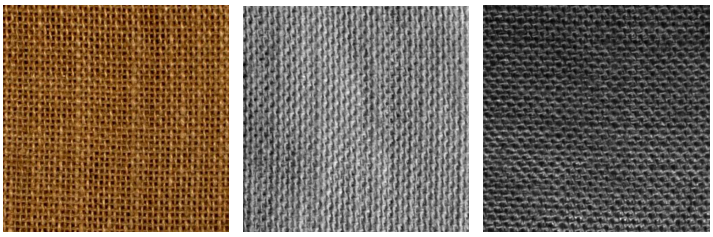


Figure 38: The albedo, P and Q gradient field images

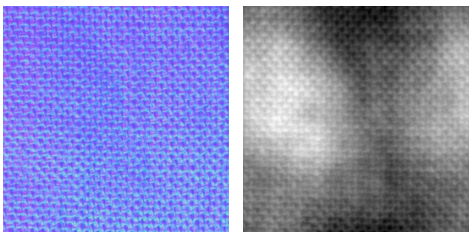


Figure 39: The final normalmap and heightmap images

The system used to implement this process was developed by Andrew Spence and Stefano Padilla on a standard Windows PC with both a professional SLR camera attached and controlled through an USB connection and a dedicated controller for the set of flash-lamps. The software for the control of the camera and flash lamps was written MFC and C++. The resulting images from the digital camera were in the raw file format for the digital camera and were converted into standard TIFF (Tagged Image File Format) files using a Linux command line utility called 'dcraw' [Coffin2000]. Conversion of the set of three *photometric images* was performed using the process described in section 4.1. Each of these stages was implemented using a Linux command line utility. In this way, the entire pipeline could be implemented using simple shell scripts. Using this system, the total amount of time required generating a pair of *albedo* and combined *normalmap* with *heightmap* images from a single textile sample would take on average 25 minutes. This time could be broken down as follows:

- Setup of the dark room, camera and automated lighting equipment – 10 minutes
- Placement of the textile sample below the digital camera and automated capture of the three *photometric images* used for *photometric stereo* – 5 minutes
- Transfer of the digital images from the camera to the controlling desktop computer and file transfer to the Linux system – 5 minutes
- Conversion of the three *photometric images* to the final *albedo* and combined *normalmap/heightmap* – 5 minutes

Much of this time can be attributed to the manual transfer of data between different systems; the transfer of *image data* from the camera to the controlling desktop computer and then to the file system of the Linux system. Because the automated camera and flash-lighting system is controlled by software written for the Microsoft Windows environment, this imposes a requirement that this operating system environment is used. While the processing time of a set of textile images is on the order of 25 minutes, this could be reduced through the use of a single integrated application that could control the acquisition of *photometric stereo* and automatically perform the conversion to the *albedo* and a *normalmap* combined with a *heightmap*. Another way in which the processing time could be reduced would be to use a smaller self contained unit in which the material sample is placed. This would eliminate the need for the use of a dark room.

In our current setup the flash-lighting system require several minutes for the capacitor to become fully charged.

4.2.1 Conclusions

In this chapter, we have described a system that utilizes *photometric stereo* for the conversion of *photometric images* to *texture data* that can be used for both *3D visualisation* and *texture retrieval*.

We can see that *photometric stereo* has the ability to capture the *micro-geometry* of textiles through the use of digital photographic methods. The use of standard digital cameras means that the *photometric stereo* technique can take advantage of features found on a modern digital camera such as high frame resolutions and macro-lens to capture the *micro-geometry* of textiles at extremely high magnifications. The process of converting the data generated from *photometric stereo* can be performed on standard desktop computer systems. These two features gives *photometric stereo* the advantage over other image acquisition techniques in that it can capture the *micro-geometry* of textiles, and that it does not require expensive custom hardware.

Having described the principles behind *photometric stereo*, we can now proceed to describe how *texture database* queries can be implemented using the resulting data. However, first we investigate how the same data can be used for *3D visualisation* purposes.

Chapter 5 – 3D Surface visualisation methods

5.1 Introduction

In Chapter 1, we identified the two research objectives for this thesis. In Chapter 2, we performed a literature survey that identified the state-of-the-art in both *texture retrieval* and *3D visualisation* techniques and we identified that while Bézier patches are suitable for the representation for *macro-geometry*, they are not suitable for the representation for the micro-geometry important to textiles. In Chapter 3 we proposed a *3D surface representation* that can be used with *texture retrieval* and *real-time 3D visualisation* techniques. In Chapter 4, we described how we used *photometric stereo* to acquire *3D surface representations*. This formed the first stage of our data representation.

In this chapter, we state our criteria for suitable *3D surface visualisation* methods then perform a survey of existing methods and finally present a summary of the candidate methods. We then describe how the *3D graphics pipeline* has been adapted to support *programmable graphics accelerator boards* using both *vertex shaders* and *fragment shaders*, before finally describing our novel method of rendering *parametric surfaces* (Bézier patches) textured with textile samples acquired using *photometric stereo* and illuminated using both *relief mapping* and *shadow mapping* with dynamic light sources to achieve *real-time* visualisation of *3D geometric objects*. This forms our core contribution of research in this thesis and the second stage of our data representation (Figure 40).

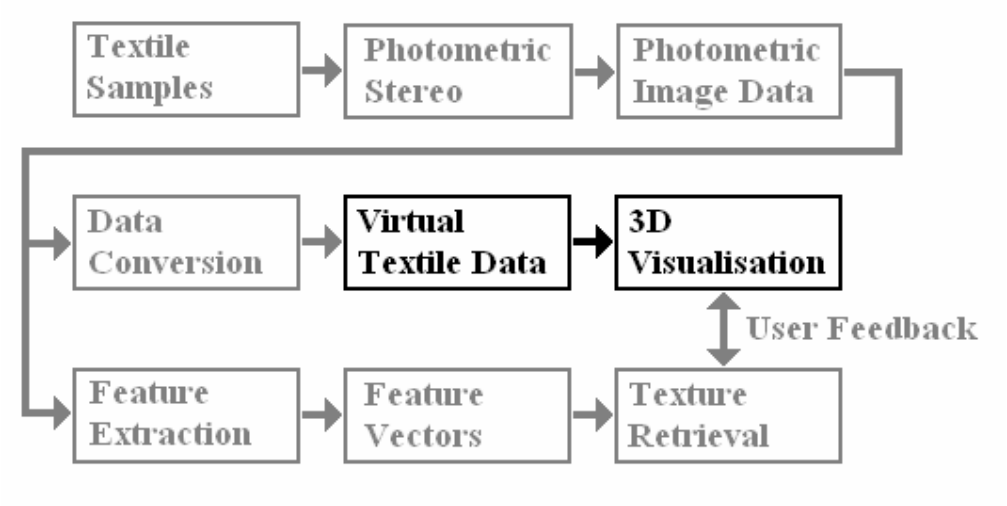


Figure 40: Stage three of the data representation

5.2 Organisation

The remainder of this chapter is organised as follows: We first present the set of criteria used to evaluate the different *rendering methods* that can deal with both the *macro-geometry* and *micro-geometry* (*3D surface descriptions*) of textile data in section 5.3 and a detailed survey of photorealistic *rendering methods* in section 5.4. Section 5.5 describes the underlying data representation common to all *visualisation* methods. Section 5.6 provides a quantitative assessment and discussion of these *visualisation* methods, with section 5.7 presenting our conclusions.

5.3 Criteria

As the main objective of this thesis is to implement *real-time 3D visualisation* of *texture data* acquired using *photometric stereo*, the choice of *rendering method* is of particular importance with regard to the presentation of the *micro-geometry* and *macro-geometry* of textile samples to the user. In order to satisfy this objective, we specify the following six criteria that must be satisfied by the candidate *rendering method*:

1. *Real-time performance*

   The user must be able to interact with light sources, rendered geometry, and camera positions in a used-friendly way as possible. All changes must occur in *real-time* at a rate not less than 15 frames per second. At frames rates lower

than this, the application will be perceived as slow and unresponsive by the users.

2. *Economic usage of memory for texture-mapping*
   The method must make efficient use of *texture memory*, especially with regard to the storage of a large collection of textile samples on the target computer. As the goal of this thesis is to be able to present a virtual catalogue of such textiles in real time, and that *programmable graphics accelerator boards* have a fixed amount of *texture memory* (currently 256 Megabytes to 2 Gigabytes), it is essential that each textile sample uses an economic amount of *texture memory*.

3. *Self-occlusion, self-shadowing and rough-edge silhouette generation*
   The method must also provide a solution to the problems of *self-occlusion*, *self-shadowing* and *rough-edge silhouette* generation.

   We define *self-occlusion* as the ability of a *rendering method* to model the ability of the raised contours of a *3D surface representation* to obscure other parts of the *3D surface representation* further away from the observer. This is different from Z-buffering in that Z-buffering only models the ability of one segment of rendered geometry to obscure other segments of geometry. The ability of a *rendering method* to model *self-occlusion* is extremely desirable if the *micro-geometry* of the *textile sample* is to be presented accurately to the user.

   We define *self-shadowing* as the ability of a *rendering method* to model the ability of raised contours of a *3D surface representation* to cast shadows on other parts of the *3D surface representation* further away from the observer. This is different from hardware *shadow-mapping* in that hardware *shadow-mapping* only models the shadows cast by different segments of rendered geometry. The ability of a *rendering method* to model *self-occlusion* is also extremely desirable if the *micro-geometry* of the *textile sample* is to be presented accurately to the user.

We define *rough-edge silhouette generation* as the ability of a *rendering method* to model the raised contours of a material when viewed from side on, even when the underlying geometry is a curved *3D geometric object*. The ability of a *rendering method* to model *rough-edge silhouette generation* is desirable but not absolutely essential to the accurate presentation of the *micro-geometry* of a *textile sample* to the user.

4. *Compatible with image data acquired from photometric stereo*
   The *rendering method* must make use of the *image data* acquired from *photometric stereo*. This *image data* includes both the *p* and *q* gradient fields as well as the derived *heightmap*.

5. *Compatible with existing 2D texture-mapping techniques*
   The *rendering method* must be able to apply existing *2D texture-mapping* techniques using current *programmable graphics accelerator boards.*

6. *Compatible with existing lighting equations models*
   The *rendering method* must be able to operate in conjunction with existing *lighting equation models* that incorporate *lighting terms* such as diffuse and specular lighting with *gloss maps*.

5.4 A detailed survey of 3D surface visualisation methods

The purpose of this section is to present a detailed survey of recent papers on the visualisation of *3D surfaces representations* with particular interest in the rendering of the *macro-texture*. Then we investigate the ability of these methods to operate in *real-time* and how the user perceives the textures presented by them.

Memory usage defines the increase in memory required to store both the *albedo* and *normalmap* textures. Typically, a single RGB texture with an *alpha channel* for *transparency* will require 4 bytes per pixel. Adding a *normalmap* with the *alpha channel* used to store the *heightmap* information will require an additional four bytes

per pixel. As a standard texture can be anywhere between 1x1 and 4096x4096 pixels in size, the total amount of *texture memory* required can range from 500 Kbytes to 16 Mbytes. While larger textures have the advantage of providing smaller detail, they have the disadvantage of reducing the number of textures that can be stored in *texture memory*. At the time of writing, current *graphics accelerator boards* have between 256 and 2 Gigabytes of *texture memory*, (although some of which may be pre-allocated to the framebuffer). The term *real-time* refers to the ability of the technique to render a complex *3D geometric model* at a responsive speed of not less than 15 frames per second.

Within the past few years, there has been a rapid advance in the capabilities of display systems for desktop computer systems. Less than a decade ago, the most demanding task for a *graphics accelerator board* was simply to accelerate framebuffer operations such as block filling and block copying, with all 3D rendering being implemented as separate application libraries or *graphics engines* running on general purpose CPU's. Consequently, such boards required less than 1 million transistors, and the only way to render complex 3D scenes was by using rasterisation based algorithms. To achieve practical ray-tracing, multi-processor systems such as supercomputers or render farms had to be used.

Today, a standard *graphics accelerator board* consists of over 330 million transistors and can render complex scenes consisting of well over 80 million triangles/second using a programmable *3D graphics pipeline* for *vertex transformation* and pixel shading. However, while this polygon rendering rate would seem to be extremely generous, this amounts to less than 1 million polygons per frame when multi-pass texturing rendering techniques [Peercy2000] are used, and even less if *fine scale surface detail* such as *micro-structure* or *micro-geometry* needs to be rendered [Koenderink1996] [Dana1999]. Following the introduction of affordable *programmable graphics accelerator boards*, many researchers have published new methods for improving the realism of rendered *bump-mapped* surfaces.

5.4.1 Texture-mapping

Texture-mapping is the simplest method in use by current *programmable graphics accelerator boards*. The advantages of this method are that memory usage is extremely efficient. However, the disadvantages of this method are that it does not provide a solution to the problem of of *self-shadowing*, *self-occlusion* or *rough-edge silhouette* generation. Thus to provide a high level of detail, a 3D geometric object must use an extremely large number of vertices.

5.4.2 Blinn Bump Mapping

Blinn bump-mapping aims to improve the *visual quality* of a texture by replacing the need for high resolution geometry with the use of a second *texture image*, the *normalmap*. The advantages of this method are that more efficient use is made of system memory. The disadvantages of this method are that that it does not provide a solution to the problem of of *self-shadowing*, *self-occlusion* or *rough-edge silhouette* generation.

5.4.3 Shell Mapping

Porumbescu, Budge, Feng, and Joy proposed an extension to *displacement mapping* that supports *3D geometric models* and procedural volume textures [Porumbescu2005]. Using this method, the 2D space defined by three vertices on the surface of the *3D geometric model* is converted into 3D by extruding the third dimension along the direction of the *surface normal* of each vertex, thus forming a thin shell. Each group of three vertices then forms a triangular prism that converts the 2D coordinate system of the *displacement map* into a 3D volume. By mapping the texture coordinates of the *displacement map* into this volume, it is possible to create small detail by adding extra geometry. The advantage of this method is that it only requires the modification of the *3D geometric object* and no complex *lighting models*, and that rendering can be achieved in a single pass. The disadvantages of this method are the memory required to store the additional geometry, and that it does not provide any solution to the problems of *self-shadowing*, *self-occlusion* or *rough-edge silhouette* generation.

5.4.4 Displacement Mapping

Cook proposed a solution to solve the problem of *rough-edge silhouette* generation and *self-shadowing* by introducing *displacement maps* [Cook1984]. Rather than just modifying the *surface normal* of a pixel before lighting, the *displacement map* modifies the actual location of each geometry vertex before the rendering process. By doing this, a polygon mesh can be deformed without creating any artifacts. While solving all three problems of *self-occlusion*, *rough-edge silhouette* generation and *self-shadowing*, this technique requires that each *3D geometric model* is rendered at a sufficiently high level of detail for each pixel of the *displacement map* to correspond to a single vertex. A compromise solution to this problem is to make use of subdivision surfaces, where a relatively low-resolution geometry model is repeatedly subdivided to gain the desired level of detail [Catmull1974], and then applying the *displacement map*. If a *displacement map* is applied to a regularly space quad grid, then it may also be referred to as a *heightmap*. Until recently, it was not possible to implement this method on *graphics accelerator boards*, as there were no instructions to support texture reading within a *vertex shader*. Becker and Max proposed a solution to unify the BRDF, *bump-mapping*, *displacement mapping* into a single algorithm [Becker1993]. Gumhold and Hüttner proposed a hardware architecture that would allow multiresolution rendering with *displacement mapping* [Gumhold1999]. Doggett and Hirche proposed a new *3D graphics pipeline* architecture that would allow adaptive tessellation of a *displacement map* using triangulated meshes [Doggett2000] [Hirche2004]. Moule and McCool also proposed an adaptive tessellation algorithm based on triangulated meshes and which is suitable for implementation on future graphics hardware [Moule2002]. The advantages of this method are that the displacement geometry can be stored as a single image or *displacement map* and that the geometry can be rendered in a single pass. The disadvantages are the 3D geometric object must have enough vertices to match the dimensions of the *displacement map*.

5.4.5 View-Dependent Displacement Mapping (VDM)

In 2003, Wang introduced *View-Dependent Displacement Mapping* as an alternative method of implementing *displacement mapping* without having the expense of texture lookup within the *vertex shader* [Wang2003]. In this method, each *normalmap* texture is replaced by a VDM texture, which has the same width and height as the original texture

but stores 32x8 viewing directions and 16 sampled *curvature* levels between –2.0 and 3.0, with interpolation being performed within those limits. The advantages of this method are that it supports *self-shadowing*, *self-occlusion* and *rough-edge silhouette* generation and that rendering of a geometric object can be achieved in a single pass. The main disadvantages of this method are that at least 64 Megabytes of *texture memory* is required to store each individual texture.

5.4.6 Horizon Mapping

In 1988, Max proposed a solution to the lack of *self-shadowing* with Blinn *bump-mapping*, by introducing the concept of a *horizon map* [Max1988]. For every pixel in the texture-map the *horizon map* stores a table of values, each of which represents the zenith angle to the horizon for a particular azimuth direction, with all directions distributed evenly around the azimuth circle. In this paper, Max recommended the use of eight directions separated by 45 degrees each. Sloan and Cohen adapted *horizon mapping* for use with *programmable graphics accelerator boards* that were state-of-the-art at the time [Sloan2000]. This method used a three pass rendering technique based on the 'NV_register_combiners' extension. The advantages of this method are that it supports *self-shadowing*, *self-occlusion* and *rough-edge silhouette* generation. However, there are several disadvantages to using this method. The first is that three rendering passes are required to render the geometric object. Another disadvantage is that representing such textures takes up a relatively large amount of memory, requiring eight samples per pixel. In addition, calculating the *horizon map* is computationally expensive, as the zenith angle for each direction of all pixels has to be calculated. Rushmeier et al, proposed a solution to this problem by describing a method of generating a *horizon map* directly from eight captured images [Rushmeier2001].

5.4.7 Parallax Mapping

Kaneko proposed a solution to the problem of the lack of *self-occlusion* within the Blinn *bump-mapping* method, by including a *heightmap* with the *normalmap* and adding a correction term that adjust the texture coordinates according to the *eye vector* and the value of the *heightmap* at the current pixel in a single iteration [Kaneko2001]. Kaneko implements this algorithm using a *fragment shader* on a current *programmable graphics accelerator board*. This method has the advantages of not requiring any additional

vertex processing or substantial increase in *texture memory* use (the only additional memory required is for the *alpha channel* of the texture to store the *heightmap*. It also has the advantage of supporting *rough-edge silhouette* generation, although for indented regions only. Welsh noticed that there was a flaw in this algorithm in the situation when the textured surface was visible at a steep angle, thus causing the texture to swim as the camera moved [Welsh2004]. Analysis revealed that the corrective term would exceed the maximum height difference of the two. Welsh's solution was to introduce an offset limit based on the height at the current pixel. However, this approach is limited to low-frequency *normalmaps*. [Mcguire2005], [Brawley2004] and [Tatarchuk2005] have each proposed improved versions in which the single iterative step in the *fragment shader* is replaced by a small iterative loop which finds the first point of intersection in the *heightmap*. This method has the advantage of improving the numerical precision of *heightmap* intersection tests, allowing for the implementation of *motion parallax*, *self-occlusion* and *self-shadowing* while still using the existing *albedo* and *normalmap texture data*. The disadvantage of this method is the problem noticed by Welsh, but solved through the use of *relief texture-mapping*.

5.4.8 Relief Texture-mapping

As an alternative method to *parallax mapping*, Oliveira proposed the method of *relief texture-mapping* in his PhD thesis [Olivieria2000]. However, due to the limited functionality of *programmable graphics accelerator boards* at this time, it was only possible to implement this algorithm in software. In the same year, Oliveria Bishop, and McAllister extended this method to run on first generation acceleration boards [Oliveria2000a][Oliveria200b]. This method still required a pre-processing stage; the first stage pre-warped the texture by the *heightmap* to take account of the direction of the *eye vector*. The second stage simply rendered the texture as a standard polygon. Fujita extended this algorithm to run on *programmable graphics accelerator boards* using the OpenGL extensions: GL_NV_register_combiner and GL_NV_texture_shader [Fujita2002]. Oliveria, Policarpo and Comba later adapted their method of *relief-mapping* to operate on arbitrary polygon surfaces [Policarpo2005], quadric surfaces [Oliveria2005] and multiple level depth maps [Policarpo2006]. One advantage of this method is that it supports *self-shadowing*, *self-occlusion* and *rough-edge silhouette* generation. Another advent is that efficient use is made of *texture memory* in that only

two *texture images* are required; the *albedo* image and the combined *normalmap* and *heightmap*, the latter of which is stored alongside the red, green and blue channels of the *texture image*. The minor disadvantage to this method is processing each individual pixel is more computationally expensive that a simple bump-mapped texture.

5.4.9 Sphere Mapping

To eliminate the accuracy problems caused by *linear searching*, Hart proposed a solution using distance functions to implement *sphere mapping* [Hart1996]. Originally intended for ray-tracing systems this technique has been adapted for use with present day *programmable graphics accelerator boards* [Donnelly2005]. The main disadvantage of *relief-mapping* is that the *linear search* stage can occasionally miss small detail on *heightmap* data, particularly the peaks of narrow ridges (Figure 41). *Sphere mapping* provides a solution to this problem by replacing the *linear search* with a distance function that returns the distance to the nearest point on the surface for every sample point in *texture space*. For optimum performance, the distance function is stored as a three-dimensional texture sized according to the width, height and depth of the *texture space*, with the depth of *texture space* based upon the precision required. Determining the intersection point between an *eye vector* and the surface starting from a point at the top of the *texture space* simply involves reading the distance texture, moving along the *eye vector* by the returned distance, and stopping once a distance of zero is returned. The advantages of *sphere mapping* are that it is efficient, and accurate, supporting *self-shadowing*, *self-occlusion* and *rough-edge silhouette* generation. The main disadvantage of *sphere mapping* is that the memory requirements range from five to sixteen times as much as a standard *bump-mapped* or *relief-mapped* texture.

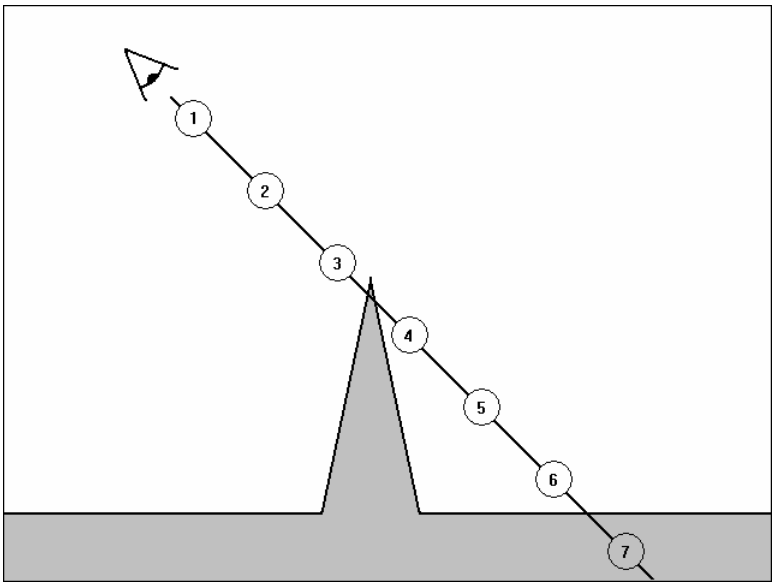


Figure 41: Example of linear search miss with relief-mapping

5.4.10 Summary

We provide a summary list of the nine candidate methods in Table 6 and describe the chosen methods in detail in the next section. As all of these methods modify either the vertices of the *3D geometry object* or the texture coordinates used to render a particular pixel, all methods are capable of supporting various *lighting equation* terms such as *gloss maps* and *transparency maps*.

As mentioned earlier, the critieria for the desired visualisation method is that it should operate in *real-time*, should have efficient memory usage and should support *edge-silhouettes*, *self-shadowing* and *self-occlusion*. From this table it can be seen that only *relief mapping* and *parallax mapping* match our criteria. However, as *parallax mapping* is essentially a special case of *relief-mapping*, we conclude that the *relief mapping* is the only method that matches our criteria.

Method	**Edge Silhouette**	**Self Shadowing**	**Self-Occlusion / Parallax**	**Memory Usage**	**Real-Time**
Texture-mapping	No	No	No	x1	Yes
Blinn bump mapping	No	No	No	x2	Yes
Shell mapping	Yes	Yes	Yes	Not	No

				known	
Displacement mapping	Yes	No	No	x2	No
View-Dependent Displacement Mapping	Yes	Yes	Yes	x64	Yes
Horizon mapping	Yes	Yes	Yes	x8	Yes
Parallax mapping	Yes	Yes	Yes	x2	Yes
Relief-mapping	Yes	Yes	Yes	x2	Yes
Sphere mapping	Yes	Yes	Yes	x5 to x16	Yes

Table 6: Summary of the candidate nine visualisation methods

5.5 The 3D Graphics Pipeline

In the canonical *3D graphics pipeline* used to render polygon geometry (Figure 42), the complete pipeline consists of four stages; (1) *Vertex transformation*, (2) Lighting, (3), Clipping and (4) rasterisation. *Vertex transformation* involves the transformation of all polygon mesh vertices relative to the combined camera and model scene positions. These are represented as two 4x4 matrices the first to represent the final transformation in *camera space*, and the second to represent the required *perspective projection* by the camera. The lighting stage applies the selected illumination model to the vertex data (flat shading, Gouraud shading, Phong shading with the option of *texture-mapping*). The clipping stage discards all vertex geometry that is outside the cube (-½, -½, -½) – (½, ½, ½). The rasterisation stage is used to scan-line render the geometry. Because of this layered structure, an implementation is free to use either software to hardware to define each stage. The sample OpenGL implementation exists entirely in software running on the host CPU. With early *graphics accelerator boards*, the transformation, lighting and clipping (TLC) stages run on the host CPU, while the rasterisation stage is implemented in hardware as a separate *graphics accelerator board*. Later *graphics accelerator boards* implemented the complete *3D graphics pipeline* as fixed functions in hardware [Deering1988] [Mccormack1998] [Deering2002].

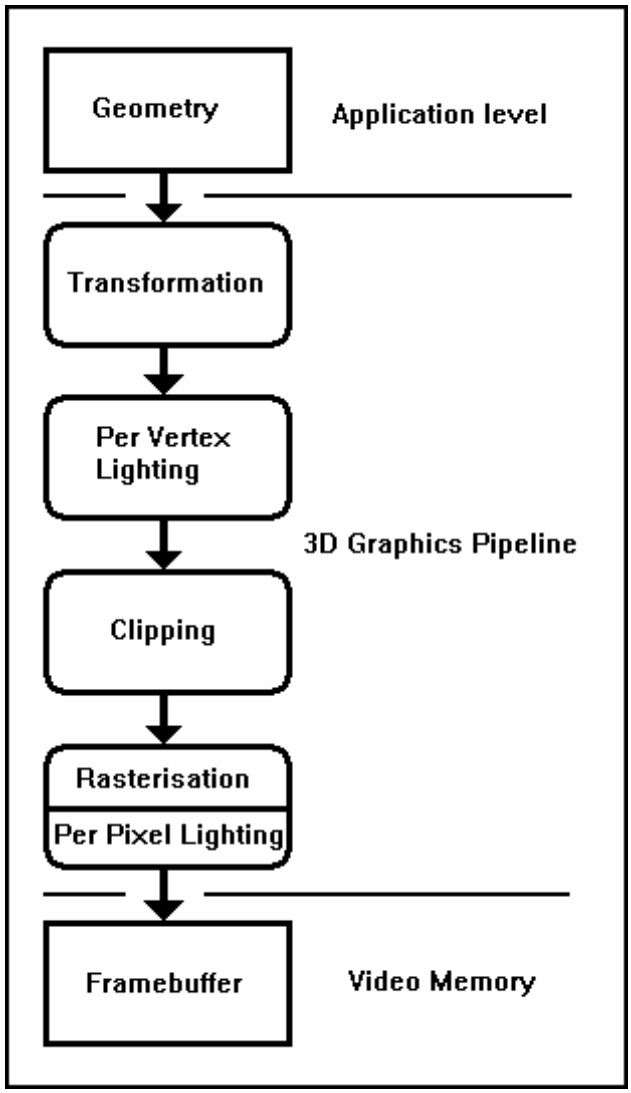


Figure 42: Classic 3D graphics pipeline

5.5.1 Vertex and Fragment shaders

A programmable *3D graphics pipeline* differs from the fixed functionality *3D graphics pipeline* in that both the transformation and lighting stages are replaced with a user defined program or "shader", both of which are executed directly on the graphics board [Whitted1982]. A *vertex shader* replaces the transformation and per-vertex lighting stages, while a *fragment shader* replaces the per-pixel lighting calculations (Figure 43).

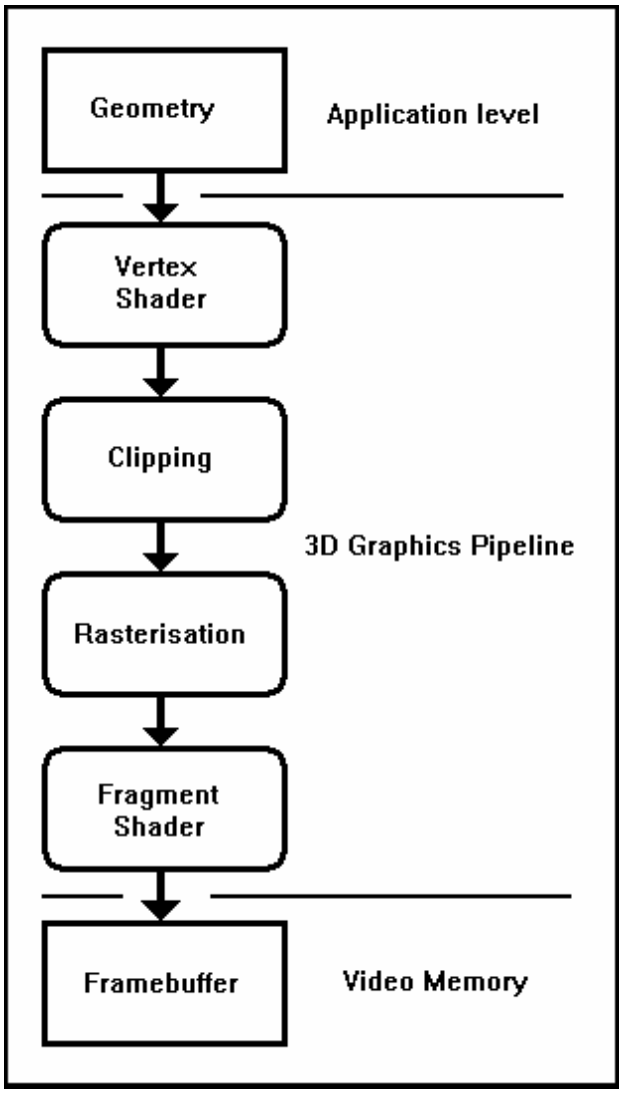


Figure 43: Programmable 3D Graphics Pipeline

5.5.2 The Tangent space coordinate system

In order to implement per-pixel lighting calculations for every point on a *3D geometric model*, we need to know several pieces of information. These include the directions of the *light source* and camera transformed into the *local coordinate system* of that point. To achieve this, it is necessary to know the *tangent space* of that point. The *tangent space* consists of three distinct vectors which form a three dimensional *local coordinate system*. These are (1) the *tangent vector*, (2) the *bi-normal vector* and (3) the *surface normal*. The *tangent vector* is a three-dimensional unit vector that defines the tangent of the surface in the direction of the first *parametric coordinate (u)*, while the *bi-normal*

vector is also a three-dimensional unit vector, which defines the tangent of the surface in the direction of the second *parametric coordinate* (*v)*. The *surface normal* is also a three-dimensional unit vector that is perpendicular to both the *tangent vector* and *bi-normal vector* (Figure 44). As all three vectors are perpendicular to each other, it is possible to calculate the *surface normal* from the cross product of the *tangent vector* and *bi-normal vector*. The *tangent space vectors* differ from the *gradient fields* p and q, in that the *gradient field* values are scalar quantities representing the two gradients at a particular point in the surface, which have no constraint on value, while the *tangent space vectors*, by definition of a unit vector, have elements which are constrained to the range -1 to +1, and the magnitude must always be equal to 1.0 exactly.

Calculation of these vectors for each vertex can be performed both during the creation of the geometry and at run-time. We use the OpenGL API to render the scene by sending the vertex, the *tangent vector*, *bi-normal vector* and *surface normal vector* to the *vertex shader*.

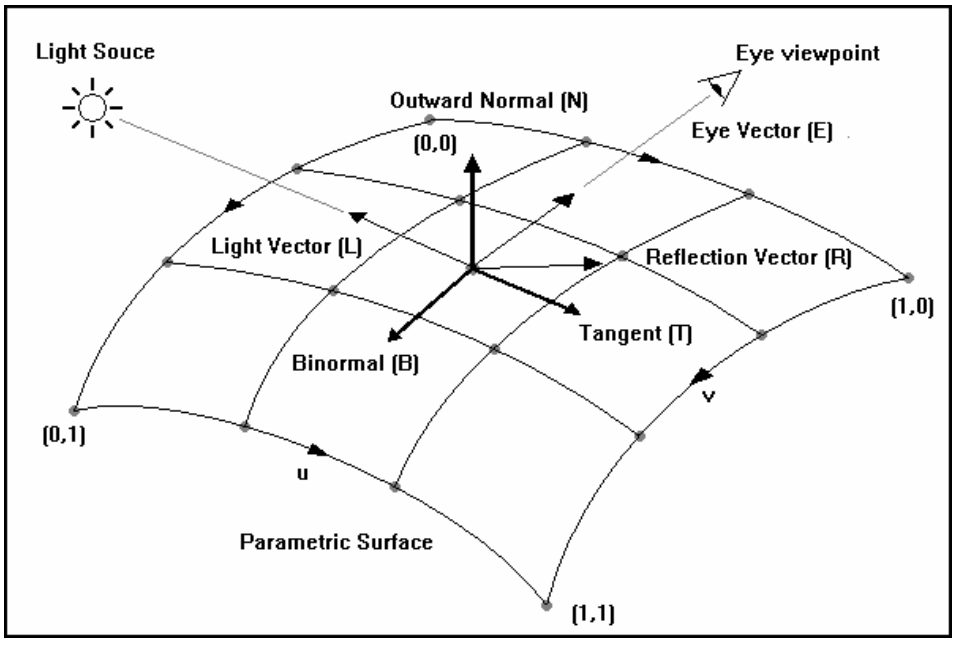


Figure 44: The Tangent Space System

When combined together the *tangent space* vectors and the coordinates of the current point in object space form a 4x4 *transformation matrix* (5.5.2.1).

$$M_{\tan gentspace} = \begin{bmatrix} t_x & b_x & n_x & p_x \\ t_t & b_y & n_y & p_y \\ t_z & b_z & n_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (5.5.2.1)$$

Where $M_{\tan gentspace}$ is the *tangent space* matrix

$\mathbf{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$ is the tangent vector,

$\mathbf{b} = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}$ is the bi-normal vector,

$\mathbf{n} = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}$ is the *surface normal* vector

and $\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$ is the *world space* coordinate of the point.

There are two ways of calculating the *tangent space* for *3D geometric models*. For rigid polygon mesh models, the *tangent space* of each vertex can be pre-calculated through the analysis of the edges forming each vertex. With higher-order surfaces such as NURBS and Bézier patch models, it is possible to perform the calculation of the *tangent space vectors* at the same time as each point of the surface. This is the approach taken for this thesis. The purpose of the *vertex shader* is thus to perform the following operations:

- Transform the vertices according to the current *model space*, *camera space* and *camera projection space* matrices
- Transform the texture coordinates according to the current texture matrix
- Transform the *world space* light positions and eye positions *into tangent space*
- Transform the vertex into light projection *shadow space* to implement *shadow mapping*

- Transform the vertex into *light projection texture* coordinates for *projective lighting*

To achieve the goal of transforming both the positions of the viewpoint and light source into *tangent space*, we combine the *tangent space* matrix, the *model space* and *camera space* matrices together. For light sources, the current direction of the light source is also calculated. For a point light source, this will vary from vertex to vertex while for directional light sources this will remain constant. These coordinates interpolated during the rasterisation stage to generate coordinates for the *lighting model* implemented by the *fragment shader*.

5.5.3 Representation of geometric models using Bézier surfaces

In the previous sections, we have described how different *rendering methods* can be used to present the *3D surface representation* to the user in *real-time*. In this section, we describe the implementation of the *geometric models* used to implement *real-time* per-pixel *bump-mapping*. To represent solid geometry, there are two possible representation methods. The first method consists of representing the solid geometry purely as lists of vertices and polygons. Each vertex is then composed of a *geometric point*, and a *tangent space* composed of a *binormal vector*, *tangent vector* and *surface normal*. Calculation of the *tangent space* data requires a detailed analysis of the connection data related to every vertex, edge and polygon. The advantages of this method are that rendering simply involves writing the geometry to the *programmable graphics accelerator board*, while the disadvantages are that calculating the *tangent space* information requires considerable analysis of the topology of the *3D geometric model*.

The second method to make use of high-order *parametric curves* and *parametric surfaces* such as NURBS [Farin2001] [Gouraud1971] or Bézier surfaces [Bézier1966] [Bézier1967] [Bézier1968] [Bézier1974]. The principle of each representation method follows a similar approach. Instead of specifying every *geometric point*, a *parametric curve* is represented by a small number of *control points* combined together using *parametric coordinates* to evaluate individual *geometric points* on the curve. This technique can be extended into two dimensions to form *parametric surfaces*. A collection of such *parametric surfaces* can be used to form a *3D geometric model*. As

well as being able to calculate individual *geometric points*, the use of *parametric coordinates* allows the *tangent space* and *texture coordinate* data to be calculated directly. This has the advantaged of eliminating the need for a detailed analysis of the connectivity of the shape, and reducing the memory requirements of storing the *3D geometric model*.

A two-dimensional matrix known as the *basis matrix* governs the shape of the curve that results from a given set of *control points*. The degree of the curve determines the number of *control points* required. For a curve of degree N, $N+1$ *control points* are required. Consequently, the *basis matrix* is also a square matrix of dimension $(N+1)\times(N+1)$. We define the mathematical relationship between the *geometric point*, *basis matrix*, *parametric coordinate* and *control points* as follows (5.5.3.1) (5.5.3.2).

$$\mathbf{p} = U.M_{basis}.C \qquad (5.5.3.1)$$

Where: $\mathbf{p}$ is the geometric point

U is the parametric basis vector

M_{basis} is the *basis matrix*,

and C is the set of *control points*

$$F(u) = \sum_{i=0}^{n} u^i (1-u)^{n-i} \frac{n!}{i!(n-i)!} c_i \qquad (5.5.3.2)$$

Where: $F(u)$ is the Bézier curve

u is the *parametric coordinate*

n is the degree of the curve

and c_i is the set of *control points*

Parametric curves can be of any degree, ranging from zero upwards. However, for low degrees (<3) there is less control over the shape of the curve, while for high degrees (>3) the large number of *control points* makes controlling the local shape of the curve difficult. Thus, spline curves of degree three or cubic curves are the most popular choice (5.5.3.3). Examples of degree three curves include the Ball spline, Beta spline, Bézier

spline, B-spline, Catmull-Rom spline, Cardinal spline, Cubic spline, Hermite spline, Kochanek-Bartels spline, Overhauser spline, Tau spline and Timmer spline curves, with each curve having a unique *basis matrix*. For curves such as the Beta spline and Kochanek-Bartels spline, it is possible to modify the *basis matrix* using additional variables that control attributes such as bias, tension and continuity. For NURBS curves, additional data values known as weights provide the user with greater control over the shape of the surface. For this thesis, we chose the Bézier curve of degree three to represent all *3D geometric models* (5.5.3.4). We present visual examples of cubic Bézier curves in (Figure 45).

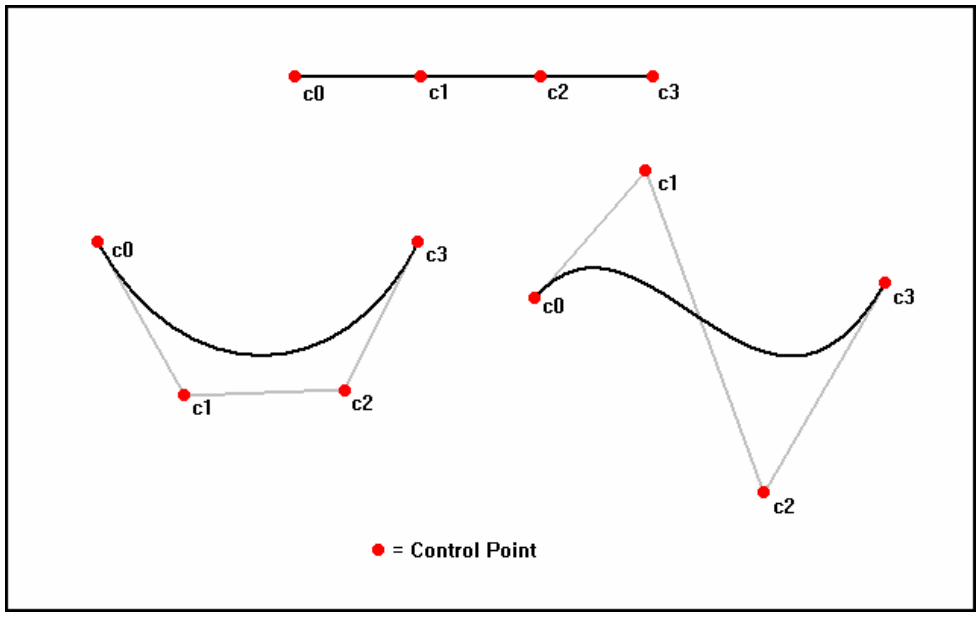


Figure 45 : Sample cubic Bézier curves

$$p(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \qquad (5.5.3.3)$$

$$p'(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 3b_{00} & 3b_{01} & 3b_{02} & 3b_{03} \\ 2b_{10} & 2b_{11} & 2b_{12} & 2b_{13} \\ b_{20} & b_{21} & b_{22} & b_{33} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Where: $p(u)$ is the spline curve,

$p'(u)$ is the first derivative of the spline curve,

u is the *parametric coordinate*,

and b_{nm} are the coefficients of the *basis matrix*

$$p(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad (5.5.3.4)$$

$$p'(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ -3 & 9 & -9 & 3 \\ 6 & -12 & 6 & 0 \\ -3 & 3 & 0 & 0 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Where: $p(u)$ is the spline curve,

p’(u) is the first derivative of the spline curve

u is the *parametric coordinate*,

and c_n are the *control points*.

To evaluate more than one *geometric point* on a *parametric curve*, it is more efficient to precalculate the product of the *basis matrix* and the set of *control points*, to (or the *geometry vector)*, and then multiply the *geometry vector* with the row vector derived from the various powers of the *parametric coordinate* for each required *geometric point.* As another optimization, forward differencing techniques help minimize the number of multiplication operations required to evaluate each point. These optimization techniques are also applicable when calculating the *tangent vector* of the curve. The *parametric surface* extends this process to two dimensions by representing the *3D geometric model* by a set of *control points* arranged in a triangulated or rectangular *control net* (Figure 46). For a rectangular *control net*, two *parametric coordinates* locate each point on the *parametric surface* (rectilinear coordinate system) while for a

triangulated *control net* three *parametric coordinates* locate each point on the *parametric surface* (barycentric coordinate system). Because at least two *parametric coordinates* are used, this also allows the calculation of the partial derivatives (or gradient values) of any *geometric point* on the surface. Since the cross product of the two gradient values produces the *surface normal*, this allows the complete *tangent space* of each *geometric point* on the *parametric surface* to be calculated. For a cubic Bézier rectangle, sixteen *control points* are required, while for a cubic Bézier triangle ten *control points* are required. We provide the general equation of the Bézier rectangle in (5.5.3.5), and provide the general equation of the Bézier triangle in (5.5.3.6).

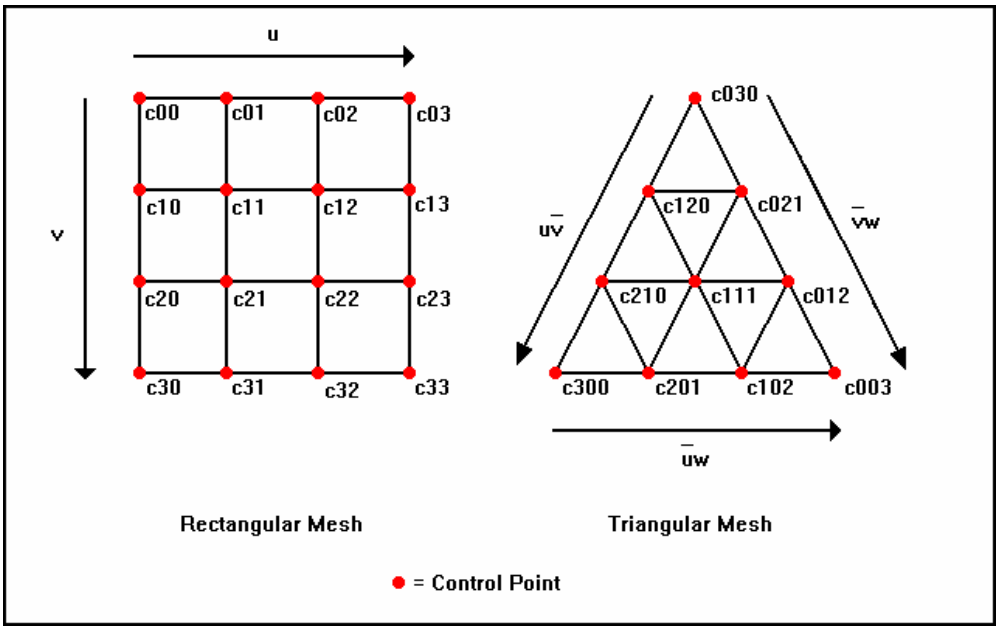


Figure 46: Control nets for rectangular and triangulated cubic Bézier surfaces

$$F(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} u^i (1-u)^{n-i} v^n (1-v)^{n-j} \frac{n!m!}{i!\,j!(n-i)!(m-j)!} c_{ij} \quad (5.5.3.5)$$

Where: $F(u,v)$ is the Bézier rectangle

(u,v) are *parametric coordinates*

n is the degree of the Bézier rectangle for u

m is the degree of the Bézier rectangle for v

and c_{ij} are the *control points* for the Bézier rectangle

$$F(u,v,w) = \sum_{\substack{i,j,k>=0 \\ i+j+k=n}} u^i v^j w^k \frac{n!}{i!\,j!k!} c_{ijk} \quad (5.5.3.6)$$

Where: $F(u,v,w)$ is the Bézier triangle function

(u,v,w) are *parametric coordinates*

n is the degree of the Bézier triangle

and c_{ijk} are the *control points* for the Bézier triangle

To evaluate a single *geometric point* on the surface of a Bézier rectangle, we use the general equation (5.5.3.7). To evaluate the *tangent vector* and the *binormal vector*, we use the partial derivatives (5.5.3.8) and (5.5.3.9), with the *surface normal* being derived from the cross product of the two vectors. Two methods for generating texture coordinates exist. The first method derives each texture coordinate directly from the *parametric coordinates*, while the second method derives each texture coordinate from the arc lengths the tangent and binormal curves.

$$
\begin{aligned}
F(u,v) = {} & (1-u)^3(1-v)^3.1.c_{00} \\
& + u(1-u)^2(1-v)^3.3.c_{01} \\
& + u^2(1-u)(1-v)^3.3.c_{02} \\
& + u^3(1-v)^3.1.c_{03} \\
& + (1-u)^3 v(1-v)^2.3.c_{10} \\
& + u(1-u)^2 v(1-v)^2.9.c_{11} \\
& + u^2(1-u)v(1-v)^2.9.c_{12} \\
& + u^3 v(1-v)^2.3.c_{13} \\
& + (1-u)^3 v^2(1-v).3.c_{20} \\
& + u.(1-u)^2 v^2(1-v).9.c_{21} \\
& + u^2(1-u)v^2(1-v).9.c_{22} \\
& + u^3 v^2(1-v).3.c_{23} \\
& + (1-u)^3 v^3.1.c_{30} \\
& + u.(1-u)^2 v^3.3.c_{31} \\
& + u^2(1-u)v^3.3.c_{32} \\
& + u^3 v^3.1.c_{33}
\end{aligned}
\qquad (5.5.3.7)
$$

Where: $F(u,v)$ is the Bézier rectangle

(u,v) are *parametric coordinates*

and c_{ij} are the *control points* of the Bézier rectangle

$$
\begin{aligned}
\frac{\partial F(u,v)}{\partial u} = {} & 3(1-u)^2(1-v)^3.1.c_{00} \\
& + (1-4u+3u^2)(1-v)^3.3.c_{01} \\
& + u(2-3u)(1-v)^3.3.c_{02} \\
& + 3u^2(1-v)^3.1.c_{03} \\
& + 3(1-u)^3 v(1-v)^2.3.c_{10} \\
& + (1-4u+3u^2)v(1-v)^2.9.c_{11} \\
& + u(2-3u)v(1-v)^2.9.c_{12} \\
& + 3u^2 v(1-v)^2.3.c_{13} \\
& + 3(1-u)^2 v^2(1-v).3.c_{20} \\
& + (1-4u+3u^2)v^2(1-v).9.c_{21} \\
& + u(2-3u)v^2(1-v).9.c_{22} \\
& + 3u^2 v^2(1-v).3.c_{23} \\
& + 3(1-u)^2 v^3.1.c_{30} \\
& + (1-4u+3u^2)v^3.3.c_{31} \\
& + u(2-3u)v^3.3.c_{32} \\
& + 3u^2 v^3.1.c_{33}
\end{aligned}
\qquad (5.5.3.8)
$$

Where: $F(u,v)$ is the Bézier rectangle function

(u,v) are *parametric coordinates*

and c_{ij} are the *control points*

$$\begin{aligned}\frac{\partial F(u,v)}{\partial v} = {} & 3(1-u)^3(1-v)^2.1.c_{00} \\ & +3u(1-u)^2(1-v)^2.3.c_{01} \\ & +3u^2(1-u)(1-v)^2.3.c_{02} \\ & +3u^3(1-v)^2.1.c_{03} \\ & +(1-u)^3(1-4v+3v^2).3.c_{10} \\ & +u(1-u)^2(1-4v+3v^2).9.c_{11} \\ & +u^2(1-u)(1-4v+3v^2).9.c_{12} \\ & +u^3(1-4v+3v^2).3.c_{13} \\ & +(1-u)^3v(2-3v).3.c_{20} \\ & +u(1-u)^2v(2-3v).9.c_{21} \\ & +u^2(1-u)v(2-3v).9.c_{22} \\ & +u^3v(2-3v).3.c_{23} \\ & +(1-u)^3v^3.1.c_{30} \\ & +u(1-u)^2v^3.3.c_{31} \\ & +u^2(1-u)v^3.3.c_{32} \\ & +u^3v^3.1.c_{33}\end{aligned} \qquad (5.5.3.9)$$

Where: $F(u,v)$ is the Bézier rectangle function

(u,v) are *parametric coordinates*

and c_{ij} are the *control points*

To evaluate a single *geometric point* on the surface of a cubic Bézier triangle, we use an equation similar to that of the Bézier rectangle (5.5.3.10). The *tangent space* (*tangent vector*, *binormal vector* and *surface normal*) system for the *geometric point* is achieved by evaluating the partial derivatives for each pair of barycentric coordinates (5.5.3.11), (5.5.3.12) and (5.5.3.13). Calculating the cross product of any two partial derivatives will generate *the surface normal.* As with the Bézier rectangle, two methods for generating texture coordinates exist. The first method derives each texture coordinate directly from the *parametric coordinates*, while the second method derives each texture coordinate from the arc lengths the tangent and binormal curves.

$$
\begin{aligned}
F(u,v,w) = {} & (v^3).1.c_{030} \\
& + (v^2 w).3.c_{120} \\
& + (uv^2).3.c_{021} \\
& + (vw^2).3.c_{210} \\
& + (uvw).6.c_{111} \\
& + (u^2 v).3.c_{012} \\
& + (w^3).1.c_{300} \\
& + (uw^2).3.c_{201} \\
& + (u^2 w).3.c_{102} \\
& + (u^3).1.c_{003}
\end{aligned}
\qquad (5.5.3.10)
$$

Where: $F(u,v,w)$ is the Bézier triangle function

(u,v,w) are *parametric coordinates*

and c_{ijk} are the *control points*

$$
\begin{aligned}
\frac{\partial F(u,v,w)}{\partial u\bar{v}} = {} & (v^2).-3.c_{030} \\
& + (vw).-6.c_{120} \\
& + (-2uv + v^2) \quad .3.c_{021} \\
& + (w^2).-3.c_{210} \\
& + (vw - uw) \quad .6.c_{111} \\
& + (u^2 + 2uv).-3.c_{012} \\
& + (w^2) \quad .3.c_{201} \\
& + (uw) \quad .6.c_{102} \\
& + (u^2) \quad .3.c_{003}
\end{aligned}
\qquad (5.5.3.11)
$$

Where: $F(u,v,w)$ is the cubic Bézier triangle function

(u,v,w) are *parametric coordinates*

and c_{ijk} are the *control points*

$$\frac{\partial F(u,v,w)}{\partial \overline{v}w} = \begin{array}{r} (v^{2)} \,.-3.c_{030} \\ +(2vw+v^{2)} \quad .3.c_{120} \\ +(uv).-6.c_{021} \\ +(w^2-2vw).-3.c_{210} \\ +(uw+uv).-6.c_{111} \\ +(u^2).-3.c_{012} \\ +(w^2) \quad .3.c_{300} \\ +(uw) \quad .6.c_{201} \\ +(u^2) \quad .3.c_{102} \end{array} \qquad (5.5.3.12)$$

Where: $F(u,v,w)$ is the cubic Bézier triangle function

(u,v,w) are *parametric coordinates*

and c_{ijk} are the *control points*

$$\frac{\partial F(u,v,w)}{\partial u\overline{w}} = \begin{array}{r} (v^2).-3.c_{120} \\ +(v^{2)} \quad .3.c_{021} \\ +(v.w).-6.c_{210} \\ +(v.w-u.v) \quad .6.c_{111} \\ +(u.v) \quad .6.c_{012} \\ +(w^2).-3.c_{300} \\ +(w^2-2uw) \quad .3.c_{201} \\ +(2uw-u^2) \quad .3.c_{102} \\ +(u^2) \quad .3.c_{003} \end{array} \qquad (5.5.3.13)$$

Where: $F(u,v,w)$ is the cubic Bézier triangle function

(u,v,w) are *parametric coordinates*

and c_{ijk} are the *control points*

Visualisation of a *parametric surface* is achieved in many possible ways; per-pixel subdivision [Catmull1974], *scan-line interpolation* [Whitted1978] [Lane1980] [Schweitzer1982], ray-tracing using *interval analysis* methods such as Bézier clipping or Newton's method [Toth1985], or by conversion into a polygon mesh suitable for rendering by a standard *3D graphics pipeline* (polygon tessellation) or for-ray tracing using ray-triangle intersection tests. Ray-tracing has the advantages of supporting advanced *lighting models* such as caustics as well as both reflection and refraction, but with the disadvantage that a single frame can take minutes if not hours to render. As the name suggests, per-pixel subdivision provides accurate represention but also requires large periods of processing time in order to subdivide the *control net* down to the resolution of individual pixels. Scan line conversion attempts to render the *parametric surface* pixel row by pixel row. While this method can operate in *real-time*, the cost of managing individual power terms makes it more costly than basic polygon mesh tessellation. Polygon tessellation methods have the advantage of being able to render frames in *real-time*, but with the disadvantage that not all advanced *lighting models* are available. Because of the requirement to render geometry in *real-time*, we have chosen the method of polygon tessellation for this thesis. Using this method, *parametric coordinates* spaced at regular intervals on the surface of each *parametric surface* of the *3D geometric model* are used to generate *geometric points*, which in turn are converted into triangle strips, all of which are stored as display lists within the *3D graphics pipeline*. Thus, only a single function call is required to render a complete *3D geometric model*.

5.5.4 Conclusions

In the previous section, we selected fourteen potential methods of rendering textures with *micro-texture*. We also described how to combine these methods with existing *3D visualisation* techniques to render the *macro-structure* (Bézier patches or *parametric surfaces.* In this section, we compare these *rendering methods* against the criteria for the *micro-structure* specified at the beginning of this chapter, and discarded those that fail to match our requirements. By comparing the set of criteria against the *rendering methods* described in the previous section, we can immediately reject some due to the large memory requirements (BRDF, BTF, PTM, VDM and *Sphere Mapping*). We reject the method of vertex *displacement mapping* techniques due to the inability to render

detailed geometry in *real-time*. We also reject basic *texture-mapping*, as it does not provide any representation for the *micro-structure*. In addition, because of their similarity, we consider *relief-mapping* and *parallax mapping* as a single method. According to these requirements, only the following methods satisfy both of our criteria:

- *Relief-mapping* using a combined *normalmap* and per-pixel *displacement mapping* using *heightmap*

This method is suitable for use with rendering with both polygonal geometry and with *parametric surfaces*. This is of particular importance to the *real-time 3D visualisation* of textile samples, as the very appearance (reflected light and texture) of such real-world textiles will change radically depending upon the combined orientation and curvature of individual points on the fabric, the position of the light source and the position of the camera. Thus, visualizing the appearance of a textile in *real-time* can only be done on a per-pixel basis and requires the computational power of a *programmable graphics accelerator board*. For this thesis, we choose to represent the macro-structure using Bézier surfaces due to their ability to automatically calculate the *tangent space* and *texture coordinate* for each *geometric point* on the surface of a *3D geometric object* and thus have provided a detailed explanation of the mathematical theory underlying their use. Knowledge of the *tangent space* of each vertex is necessary in order to present the *micro-structure* to the user through the use of *relief-mapping,* as the *light vector* must be transformed from the *world space* system to the *tangent space* system and the *eye vector* must be transformed from the *camera space* system to the *tangent space* system. To allow the user to view the model with as much freedom as possible, the user interface has been designed to allow the user to control the position of the model, light sources and camera independently. The operations supported include rotating the model, rotating and zooming both the camera and light-sources. All objects can be allowed to rotate automatically, to brake automatically, or to only rotate whenever the user moves the mouse. Light sources are rendered as *3D geometric objects* in order to give the user feedback as to where the light source is located and moving. The novel combination of using *parametric surfaces* (*Bézier surfaces*) to represent the *macro-structure* combined with *texture images* of textile samples acquired using *photometric stereo* to represent the *micro-structure* and illuminated using both *relief mapping* and *shadow mapping*

with dynamic light sources to achieve *real-time* visualisation of 3D geometric objects forms our core contribution of research in this thesis.

Having described the background theory behind the interactive visualisation stage of this thesis, it is the purpose of the next chapter to present the sample images from our *3D visualisation* system that combines together the rendering of the *macro-structure* using *Bézier patches* combined with *shadow-mapping* and the *micro-structure* using *relief-mapping*.

Chapter 6 –Visualisation Implementation and Results

6.1 Introduction

In Chapter 5, we stated our criteria for a suitable *3D surface visualisation* method, performed a survey of approaches and then presented a summary of the candidate methods. We then described how the *3D graphics pipeline* has been adapted to support *programmable graphics accelerator boards* using both *vertex shaders* and *fragment shaders*, before finally describing how the *macro-structure* could be defined in terms of *parametric surfaces* (Bézier patches) combined with the *micro-structure* description in the form of *normalmaps* combined with *heightmaps* and illuminated using both *relief mapping* and *shadow mapping* with dynamic light sources to achieve *real-time* visualisation of *3D geometric objects*.

This chapter describes the implementation of our *3D visualisation* system. Of particular importance is the integration of the rendering of the *macro-structure* using *Bézier patches* rendered using *shadow-mapping* and the *micro-structure rendered* using *relief-mapping*. We begin this chapter by describing the process of generating *texture data* for both the *bump-mapping* and *relief-mapping* methods, and the *lighting model* used to relight this *texture data*, then how each method is implemented using a *programmable graphics accelerator board*.

We now proceed to describe how *bump-mapping* is implemented using OpenGL, how this is extended to implement *relief-mapping* using OpenGL, and finally describing how the *lighting model* is implemented.

6.1.1 Bump-mapping

Rendering a *bump-mapped* surface using a *programmable graphics accelerator card* requires two *texture images*; the *albedo* and the *normalmap*. Each image consists of either RGB or RGBA *texture data* with the *alpha channel* free for other purposes such as a *gloss map* or a *transparency map*. In order to render a texture mapped surface with a *programmable graphic accelerator card*, the *vertex shader* is used to transform both the *eye vector* and *light vector* into *tangent space*, and the *fragment shader* simply fetches the appropriate texture elements from each of the *albedo* and *normalmap* textures, before applying the Phong *lighting model* using the *surface normal* from the *normalmap* texture.

6.1.2 Relief-mapping

In his 2005 paper, Policarpo solves the problems of *self-occlusion*, *self-shadowing* and *rough-edge silhouette* generation by the use of a *heightmap* that represents the surface depth at each pixel sample of the *normalmap.* The point of intersection between the viewer and the textured surface is determined through a *linear search* followed by a *binary search*. In his paper, Policarpo implements the algorithm as a *fragment shader* on current generation *programmable graphics accelerator boards*.

For this thesis, we use the exact method of *relief-mapping* as described in this paper. This method requires the same *albedo* and *normalmap* texture as used with standard *bump-mapping*, but also requires the use of a *heightmap*. In normal use with our application, the *heightmap* is stored in the *alpha channel* of the *normalmap* texture.

With *relief-mapping*, the *fragment shader* uses a three-stage process to determine the correct *albedo* colour and *surface normal* values rather than the single stage lookup process used by *bump-mapping*. The first stage solves the problem of *self-occlusion* by using *interval analysis* to query the *heightmap* and determine the first point of the surface that intersects the *eye vector*. The actual process of *interval analysis* with an unknown function involves two internal stages. The first step involves using a *linear search* to find the first interval in which the *eye vector* crosses the surface boundary. This is necessary, because an irregular surface with peaks and troughs may intersect

with the *eye vector* in more than one place. The second step involves using a *binary search* (such as the *bisection method*) to find the precise point of intersection within this interval. While the *normalmap* and *heightmap* data are stored in a discrete data format (ie. sample points at fixed intervals in *texture space*), the actual data returned from the texture sampling function 'texture2D' will be composed from the average of two or more texture elements, depending upon the texture *minification* and *magnification MIP-mapping* modes. *Minification* occurs when the rendered texture occupies a smaller number of pixels in the framebuffer than the original texture. *Magnification* occurs when the rendered texture occupies a larger number of pixels in the framebuffer than the original texture. These modes are described in Table 9-3 of the OpenGL Programming Guide [Shreiner2004]. By default, any ray which crosses any of the texture boundaries $(u = 0), (v = 0), (u = 1)$ or $(v = 1)$ of the texture will *wrap-around* to the opposite side if the texture repeat mode is set. This allows a *single relief-mapped* texture to repeat across the surface of a *3D geometric model*. For open *3D geometric models* (such as a simple plane), the *fragment shader* can implement *edge silhouetting* by discarding the current pixel fragment using the shader language '*discard()*' call, if the intersection of the ray is outside any of these boundaries. The second stage implements the chosen *lighting equation* and *scene-shadowing* tests as before. The third stage implements a *self-shadowing* test by looking for the first point visible along the line-of-sight vector towards the light-source. If the point matches the location of the light source, then no *self-shadowing* is occurring for the current pixel fragment, and the final fragment colour remains unchanged. Otherwise, the point is in shadow, and shaded accordingly. We present an example of this process in (Figure 47). In this example, the *eye-ray* intersects three points within the relief map. Without the *heightmap* test, a basic texture lookup would select the colour and *surface normal* at point (A). However, with *relief-mapping*, the algorithm will select the first point (1). When the *fragment shader* performs the shadow test for this point, point (4) is in the line-of-sight, and so point (1) is determined to be in shadow.

Having described the *visualisation* method, we now proceed to describe how the applications implements the rendering of geometry, and how the *fragment shader* implements the *lighting equation* for *bump-mapping* and *relief-mapping*, before describing how quantitative assessment is performed.

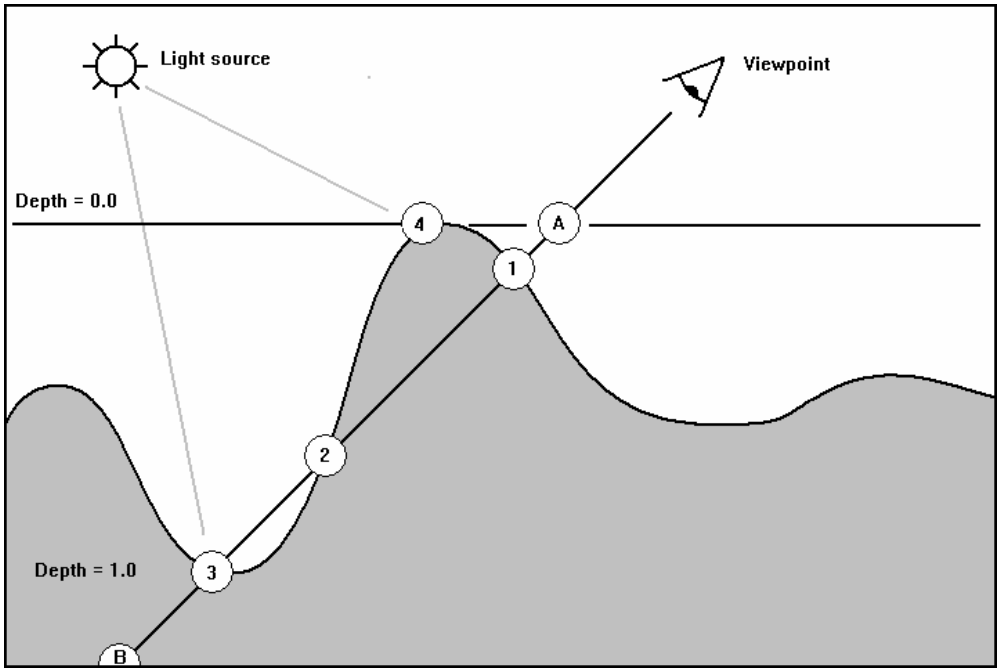


Figure 47: Example height-map query for relief-mapping.

6.1.3 Relighting the 3D surface representation

In the previous section, we described how a *programmable graphics accelerator board* may be used to render the scene in *real-time*. We also described how we acquired *gradient field* data using *photometric stereo*, and how we generated texture maps suitable for with a *graphics accelerator board* from this *gradient field* data. The goal of this section is to describe how the *3D visualisation* application combines the basic components (light sources, cameras, geometry and textures) together in order to render a photorealistic image. Fundamental to the solution of this problem is the *lighting equation*. The *lighting equation* defines the mathematical model that describes how *surface materials* receive and transmit light emitted by light sources and other *surface materials*. Gouraud was the first to propose a *lighting model* using the Lambert shading model [Lambert1760] [Gouraud1971]. Phong modified this equation to handle non-diffuse surfaces [Phong1975]. Blinn and Phong adapted this equation for use on graphics hardware [Blinn1977]. Cook and Torrrance proposed a more accurate model based on Gaussian modeling of micro-facets [Cook1982]. Since then, the *lighting equation* has evolved to model *anisotropic reflection* [Banks1994] and multi-spectral radiosity calculations [Neumann2003].

For this thesis, we selected the *Phong lighting model* [Phong1975] as the basis upon which in order to implement the *lighting model*. This *lighting model* allows for the implementation of ambient and diffuse terms to implement Lambert shading for matte surfaces, and a specular term for glossy surfaces (6.1.4.1).

$$\mathbf{l_i} = \mathbf{k_a} + \mathbf{k_d}\mathbf{f_{att}}(\mathbf{n}\cdot\mathbf{l}) + \mathbf{k_s}\mathbf{f_{att}}(\mathbf{e}\cdot\mathbf{r})^{k_{sp}} \qquad (6.1.4.1)$$

Where: $\mathbf{l_i}$ is the resulting output intensity for the pixel fragment,

$\mathbf{k_a}$ is the fraction of light emitted from the surface by ambient reflection,

$\mathbf{k_d}$ is the fraction of light emitted from the surface by diffuse reflection,

$\mathbf{k_s}$ is the fraction of light emitted from the surface by specular reflection,

k_{sp} is the specular power factor,

$\mathbf{f_{att}}$ is the fraction of light that reaches the surface due to the attenuation of light,

n is the *surface normal* (normalized direction vector perpendicular to the surface),

l is the *light vector* (normalized direction vector pointing towards the light source),

e is the *eye vector* (normalized direction vector pointing towards the viewpoint), and

r is the *light vector* reflected through the *surface normal*.

The terms $\mathbf{k}_a$, $\mathbf{k}_d$ and $\mathbf{k}_s$ and k_{sp} define the RGB ambient, diffuse, specular values and specular power terms of the current *surface material* respectively. However, Phong originally intended the *lighting model* to model untextured surfaces. To adapt the Phong model for use with *bump-mapped* textured surfaces, we have to make several modifications. We replace the ambient and diffuse terms of the *surface material* $\mathbf{k}_a$ and $\mathbf{k}_d$ with texture map references to the *albedo* (or base) texture $\mathbf{k}_\rho$, and replace the *surface normal* $\mathbf{n}$ of the surface with a texture lookup reference to the bump-map texture $\mathbf{k}_{normalmap}$. We also replace the specular term $\mathbf{k}_s$ with a texture map reference to $\mathbf{k}_\rho$, combined with a reference to the *gloss map* texture $\mathbf{k}_{gloss}$. This texture can be estimated manually, set to zero, or acquired through the combined use of *photometric stereo* with a polarizing filter and the *Fresnel reflection* model [Wolff1990]. We also replace the single power term $\mathbf{k}_{sp}$ with separate powers for each colour. We also augment the term $\mathbf{k}_s$ with local parameters $\mathbf{k}_{gloss}$. For a surface with texture coordinates (u,v), we define the relationships between the *albedo* and *normalmap* textures and the ambient, diffuse and specular terms by the expressions in (6.1.4.2).

$$\begin{aligned} \mathbf{k}_a &= \mathbf{k}_d = \mathbf{k}_\rho(u,v) \\ \mathbf{k}_s &= \mathbf{k}_{gloss}(u,v).\mathbf{k}_{specfactor} \\ \mathbf{n} &= \mathbf{k}_{normalmap}(u,v) \end{aligned} \qquad (6.1.4.2)$$

Where: $\mathbf{k}_a$ is the ambient term of the *Phong lighting model*,

$\mathbf{k}_d$ is the diffuse term of the *Phong lighting model*,

$\mathbf{k}_s$ is the specular term of the *Phong lighting model*,

$\mathbf{k}_{\mathbf{gloss}}$ is the surface *specularity map*,

$\mathbf{k}_{\mathbf{albedo}}$ is the *albedo* texture,

and $\mathbf{k}_{\mathbf{normalmap}}$ is the *normalmap* texture

We augment these terms by additional terms used to implement shadowing in order to support a completely generic texture description as described by Stürzlinger [Stürzlinger1996]. We also implement several control variables to allow the various options to be switched on and off through user control. We present listings of each *vertex shader* and *fragment shader* used by this application in "Appendix C – OpenGL vertex and fragment shaders".

To implement *shadow mapping*, we utilize the method described by Williams [Williams1978]. In this method, a *depth texture map* representing the distance of the light source to the nearest point on each visible surface of the scene is created by rendering the scene as seen from the viewpoint of the light source. To render the scene with shadows, the *fragment shader* transforms and compares the final depth value of each pixel fragment against the corresponding value in the *shadowmap*. The result of the comparison test determines whether the pixel fragment is in shadow. We present the assignment of textures to texture units within the *programmable graphics accelerator* in Table 7.

Texture	**Texture Unit**
k_{ρ}	Unit 0 – RGB
k_{gloss}	Unit 0 – Alpha
$k_{bumpmap}$	Unit 1 – RGB
$k_{heightmap}$	Unit 1 – RGB
k_{shadow}	Unit 2 – Depth
$k_{projector}$	Unit 3 – RGBA

Table 7: List of textures and their assigned texture units

The resulting *lighting model* for *texture-mapped bump-mapping/relief mapping* is thus as follows:

$$l_i = \sum_{i=1}^{n} \mathrm{L}_n(\mathbf{e}_{\mathbf{tangentspace}}) \qquad (6.1.4.3)$$

Where: l_i is the resulting output intensity for the pixel fragment,

and L_n is the fraction of light emitted from the surface from ambient, diffuse and specular reflection

We define the lighting function L for a single light source to be:

$$L = l_{ambient} + (\mathrm{L}_{\mathrm{diffuse}} + \mathrm{L}_{\mathrm{specular}}).\mathrm{rm\_shadow}(\mathbf{l}, \mathbf{e}_{\mathbf{tangentspace}}).\mathbf{l}_{\mathbf{shadow}} \qquad (6.1.4.4)$$

Where: $l_{ambient}$ is the ambient contribution of the light source,

$\mathrm{L}_{\mathrm{diffuse}}$ is the diffuse lighting contribution,

$\mathrm{L}_{\mathrm{specular}}$ is the specular lighting contribution,

$\mathbf{l}_{\mathbf{shadow}}$ is the sample point of the light source shadow texture,

and rm_shadow is the *relief-mapping* shadow function

We define the diffuse component of the lighting function to be:

$$\mathrm{L}_{\mathrm{diffuse}} = \mathbf{c}_{\mathbf{albedo}}.\mathbf{l}_{\mathbf{diffuse}}.(\mathbf{n} \cdot \mathbf{l}) \qquad (6.1.4.5)$$

Where $\mathbf{c}_{\mathbf{albedo}}$ is the *albedo* colour of the current point of the surface,

$\mathbf{l}_{\mathbf{diffuse}}$ is the contribution of the light source to diffuse lighting,

$\mathbf{n}$ is the outward normal of the current point of the surface

and $\mathbf{l}$ is the *light vector* in the local tangent space of the object

We define the specular component of the lighting function to be:

$$L_{specular} = \mathbf{c}_{\mathbf{albedo}} . \mathbf{l}_{\mathbf{specular}} . (\mathbf{e}_{\mathbf{tangentspace}} \cdot \mathbf{r})^{k_{specularpower}} . \mathbf{k}_{\mathbf{specularfactor}} \qquad (6.1.4.6)$$

Where $\mathbf{c}_{\mathbf{albedo}}$ is the *albedo* colour of the current point of the surface,

$\mathbf{l}_{\mathbf{specular}}$ is the contribution of the light source to specular lighting,

$\mathbf{n}$ is the outward normal of the current point of the surface,

$\mathbf{l}$ is the *light vector* in *tangent space*,

$\mathbf{e}_{\mathbf{tangentspace}}$ is the eye vector in *tangent space*,

$\mathbf{k}_{\mathbf{specularpower}}$ is the set of power coefficients for specular lighting,

$\mathbf{k}_{\mathbf{specularfactor}}$ is the material colour for specular lighting

and $\mathbf{r}$ is the *light vector* reflected through the *surface normal*.

However, for this thesis, we discovered that for many textile samples, we could set the contribution of the specular term to zero or close to zero, as the textile samples did not have a glossy appearance.

6.1.4 Implementing the lighting equation using vertex and fragment shaders

To implement the complete *lighting equation*, we split the task of visualizing the target geometry up into a separate rendering pass for each light source. We use custom designed *vertex shaders* and *fragment shaders* to implement each of the rendering passes. We implement multiple light sources by making use of the OpenGL blend operations. For the first layer of the first light source, the blend mode is set to replace, while for every other layer the blend mode is set to add [Shreiner2004]. We utilize the method of *shadow mapping* as described by Crow [Crow1977], Williams [Williams1978] and [Woo1990] to implement scene level shadowing. This method requires two rendering passes. In the first pass, the *shadowmap* is created by enabling the Z-buffer, disabling *texture-mapping* and rendering the light source's view of the scene. The second pass involves rendering the camera's view of the scene with both

texture-mapping and the Z-buffer enabled as usual. As rendering of each pixel on the surface occurs, the *fragment shader* transforms the *camera projection space* coordinates into *light projection space*. Next, the *fragment shader* tests the resulting coordinate against the *shadowmap*. If the depth value of the pixel fragment is less than the value in the *shadowmap*, then the corresponding point on the surface is visible to the light source and the *lighting equation* evaluated. Otherwise, if the pixel fragment is not visible to the light source, then the light source makes no diffuse or specular contribution to the final pixel colour. While our original *lighting model* was designed to support advanced lighting effects such as *environment mapping*, reflection, refraction, *chromatic aberration* and *projective lighting*, we found that these were not necessary for the visualisation system.

6.1.5 Rendering the contribution of each light source in the scene

To calculate the contribution to the scene made by each type of light source, we use a custom *vertex shader* and *fragment shader*, which implement our *lighting model*. We provided a detailed explanation of these in Appendix C.

6.1.6 Rendering the 3D geometric models

With all the shader programs implemented, it is thus possible to render both flat and curved *3D geometric models* using many different combinations of the techniques described earlier in this chapter. These combinations include flat (planar) geometry vs curved geometry, static vs. animated geometry (slow rotation, rippling wave effects, free-form deformation), *bump-mapping* vs. *relief-mapping*, hardware *shadow-mapping* vs. no *shadow-mapping*, directional light sources vs. point light sources, single light sources vs. multiple light sources, and any one of six combinations of ambient, diffuse, and specular lighting for any *3D geometric model*. However, for many of these techniques, it is obvious or has been demonstrated by previous research, that one choice will always provide a more photo-realistic appearance than the other (eg. *shadow-mapping* vs. *no shadow-mapping*). Thus for the purposes of this thesis, we make the following decisions in the setup of the visualisation application experiments. Due to the complexity of three of the chosen *3D geometric models*, we choose to keep the geometry rigid, and instead animate all the *3D geometric models* using a slow rotation method. We choose *relief-mapping* over *bump-mapping* due to the extra fine-scale

detail provided. We choose a single light source over multiple light sources for performance reasons (as individual light sources require multiple passes). We also choose hardware *shadow-mapping* over no *shadow-mapping* as there are available texture units to handle this task. We choose a *point light-source* over a *directional light source* due to the additional realism that this provides through the use of localized light intensity highlights. We also use a full *lighting model* with both an ambient and diffuse terms to relight the textile samples as realistically as possible. While we did also implement a *specular lighting* term in our *lighting model*, we found that no contribution was required in order to match the original *photometric images*, and so omitted this term during the implementation of our system. In the next chapter we present a demonstration of the visual output of our system.

6.2 Demonstration of visualisation methods

In section 6.1, we described our implementation of the visualisation system based upon our literature survey. In this chapter, we present rendered images from our visualisation system and discuss how each visual effect improves the photorealistic appearance of the textile sample. For this thesis, we chose to implement both the visual user interface and rendering libraries using the C++ programming language and OpenGL, with the goal of keeping the application multi-platform. For the Windows XP/Vista operating system we used Microsoft Visual Studio as the development environment, with Nvidia Geforce FX6800 and FX8800 programmable graphics accelerator boards as the target hardware.

6.2.1 Comparison of the visual effects

In this section, we present a selection of the samples of our *texture database* rendered using five different methods:

- *texture mapping* with no lighting
- *texture mapping* with diffuse lighting
- *bump-mapping* with diffuse lighting
- *relief-mapping* with diffuse lighting
- *relief-mapping* with diffuse lighting and shadow-mapping

We present sets of concave geometry rendered using each of these five methods below. In (Figure 48), we render the geometry using *texture mapping* with no lighting. In (Figure 49), we render the geometry using *texture mapping* combined with *diffuse lighting*. In (Figure 50), we render the geometry using *bump-mapping* combined with *diffuse lighting*. In (Figure 51), we render the geometry using *relief-mapping* combined with *diffuse lighting*. Finally, in (Figure 52), we render the geometry using *relief-mapping* and *diffuse-lighting* combined with *shadow-mapping*.

From a visual comparison of the rendered images in (Figure 48) and (Figure 49), we can be seen that the use of diffuse lighting provides the greatest improvement on the appearance of the texture. From a visual comparison of the rendered images in (Figure 49) and (Figure 50) we can see that the introduction of *bump-mapping* provides another improvement in visual appearance as there is a higher level of contrast between the lighter and darker areas of the textile sample – these are caused by the natural *curvature* between the peaks and valleys of the *micro-geometry* of the textile sample. From a comparison of the rendered images in (Figure 50) and (Figure 51), we can see that the use of relief-mapping further increases the detail of the rendered *micro-geometry* of the textile sample. Finally in (Figure 52), we can the added use of *shadow mapping* to the rendering of geometry with relief-mapping and diffuse lighting provides the highest quality of visual realism. We present an additional set of rendered images in (Figure 53), (Figure 54), (Figure 55) and (Figure 56) with textile samples applied onto complex geometry such as the Utah Teapot and a torus knot.or trefoil and rendered using *relief-mapping*, *diffuse-mapping* and *shadow-mapping* all combined together.

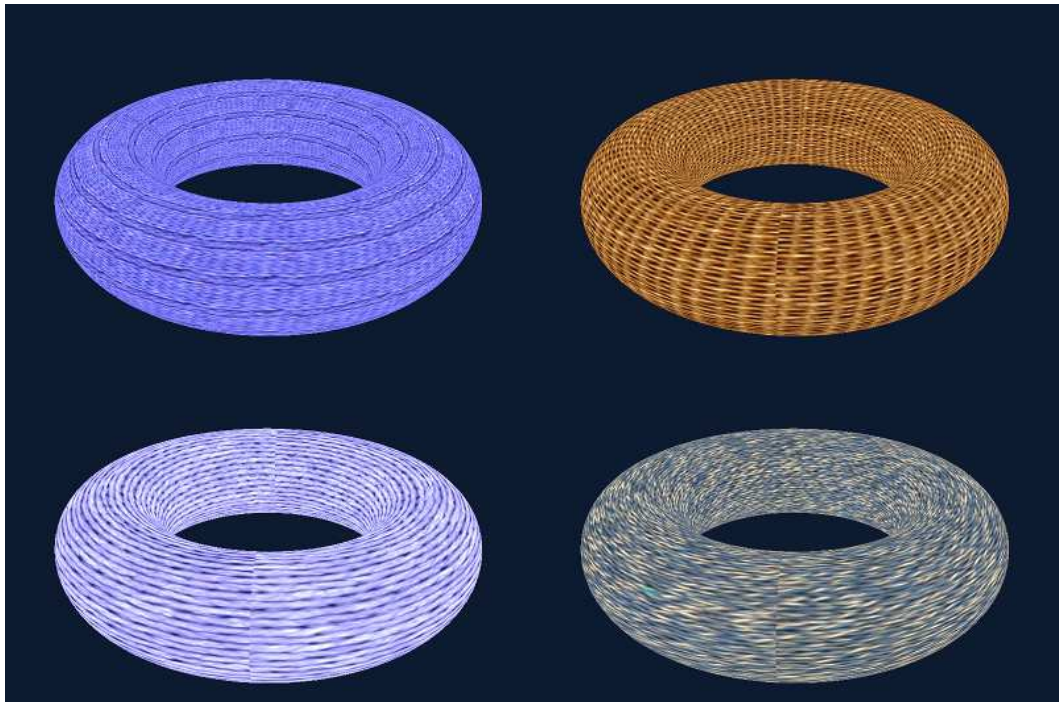


Figure 48: Geometry rendered with standard texure mapping and no lighting

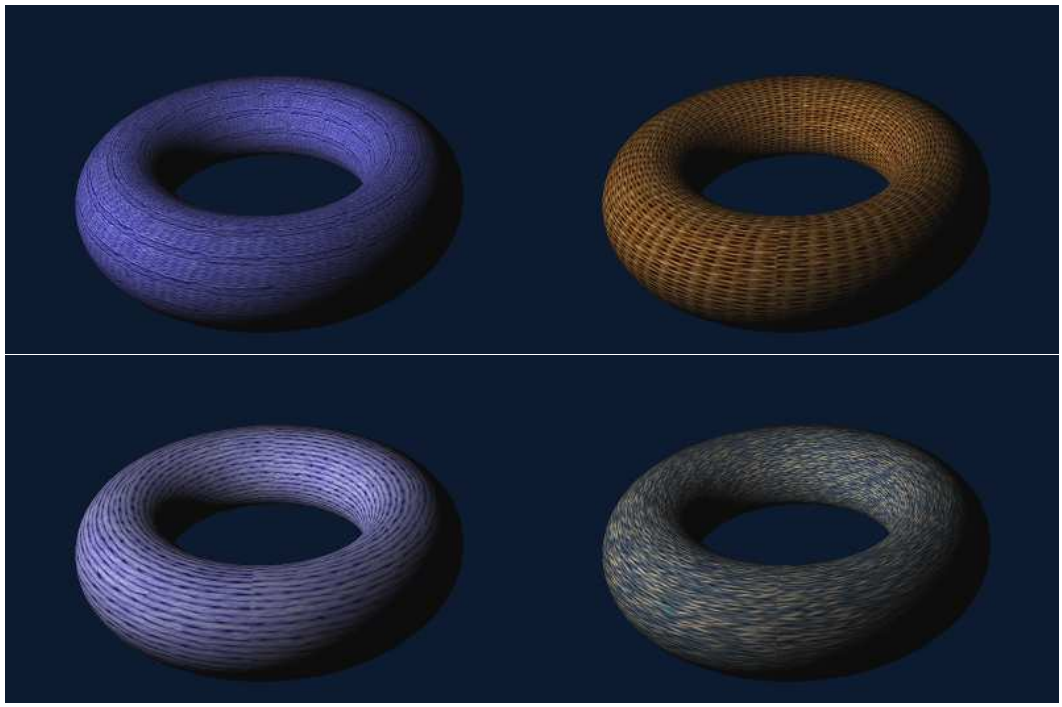


Figure 49: Geometry rendered with standard texture mapping and diffuse lighting

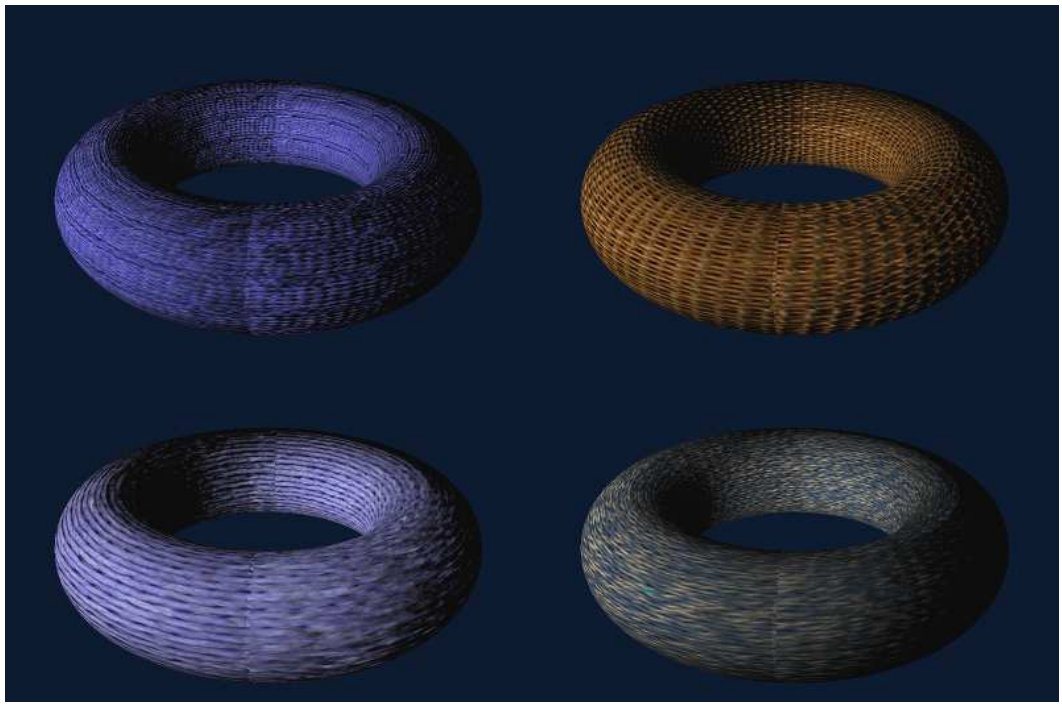


Figure 50: Geometry rendered using bump-mapping and diffuse lighting

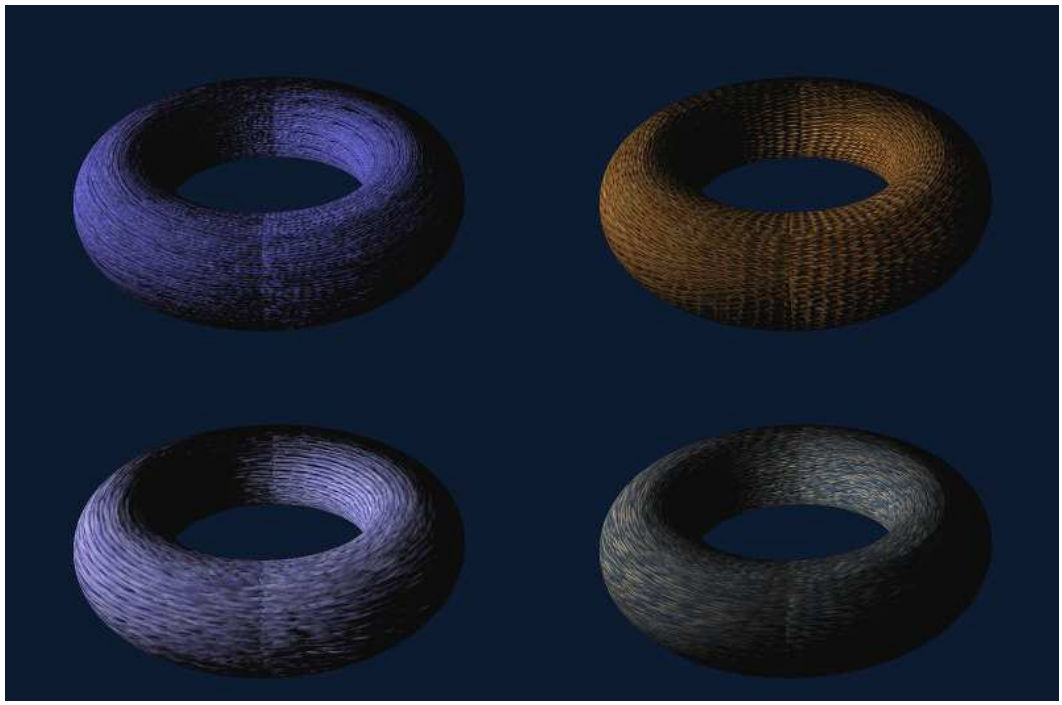


Figure 51: Geometry rendered using relief-mapping and diffuse lighting

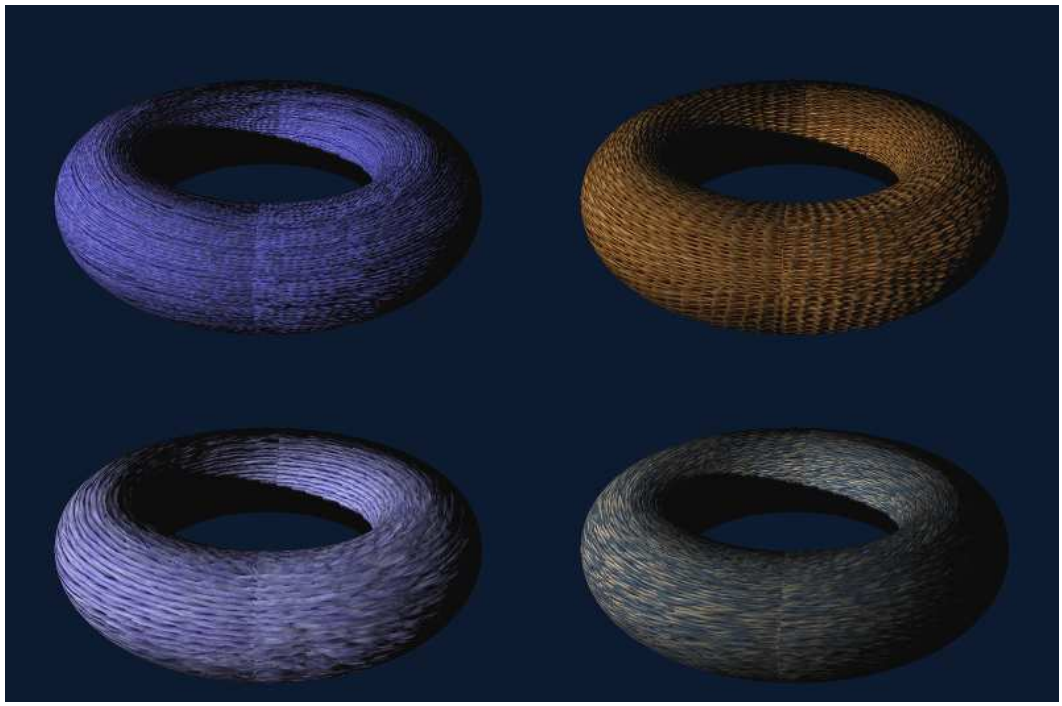


Figure 52: Geometry rendered with relief-mapping, shadows and diffuse lighting

Figure 53: Torus knot rendered with relief mapping, diffuse lighting and shadows

Figure 54: Utah Teapot rendered with relief mapping, shadows and diffuse lighting

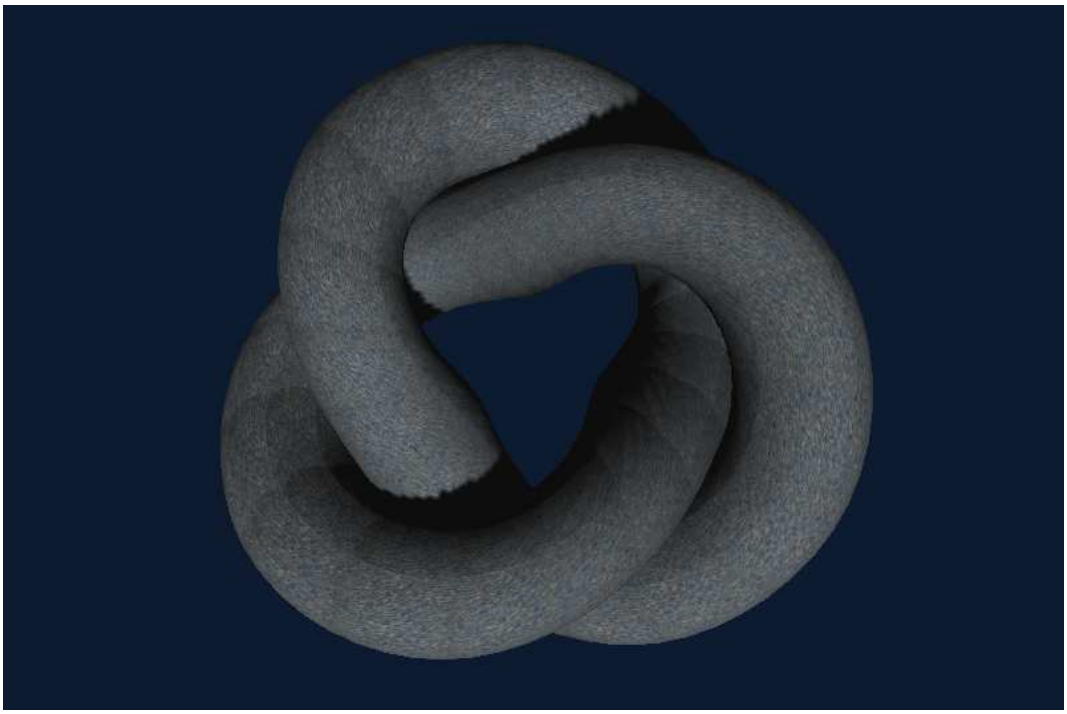


Figure 55: Torus knot rendered with relief mapping, shadows and diffuse lighting

Figure 56: Utah Teapot rendered with relief mapping, shadows and diffuse lighting

6.2.2 Conclusion

This chapter has presented images of 3D geometric objects rendered with textile samples acquired through the use of *photometric stereo* techniques. We have demonstrated that *bump-mapping* is an improvement over *texture-mapping* for the *visual quality* of textile samples, and that *relief-mapping* is a further improvement over *bump-mapping* in terms of *visual quality*. We also demonstrate that the combined use of *macro-structure* and *micro-structure rendering methods* through the use of *shadow-mapping* with both *relief-mapping* and diffuse lighting provides a further refinement to the *visual quality* of rendered textile samples. Since the main objective is to develop an economical method for the retrieval and *visualisation* of 3D surface *micro-texture* we selected the *relief-map rendering method*, which requires one photometric *albedo* image and a *relief map* (one *normalmap* combined with a *heightmap).* The *relief-mapping* method assumes that the surface is rough, and takes into account *self-shadowing* using the *light-vector* intersection tests and *self-occlusion*. This *rendering method* is compatible with current *programmable graphics accelerator boards*, with the surface

micro-texture being converted into *texture data* suitable for *real-time rendering* using a combination of *vertex shaders* and *fragment shaders*.

Chapter 7 – Texture Retrieval Methods

7.1 Introduction

In Chapter 1, we identified our three main research objectives the third objective of which is to implement *rotation invariant texture retrieval*. In Chapter 2, we identified from our survey of the literature that *rotation invariant filter banks* and *colour histograms* were promising sets of *feature vectors*. In this chapter therefore, we identify five criteria for *rotation invariant texture retrieval* and ten candidate *texture feature similarity operators,* with particular attention given to *filter banks* sensitive to the periodicities that occur within textile samples. We then compare the performance of each of the *feature vectors* using Receiver-Operator-Characteristic and Precision-Recall graphs, before presenting our conclusions for this chapter. This forms the third stage of our data representation, a pilot study into the implementation of an *information retrieval* system. As discussed in Chapter 2, the goal is not to develop a complete information retrieval system, but to focus our attention on the implementation of the feature extraction and similarity search operators rather than the user interface or user relevance feedback system.

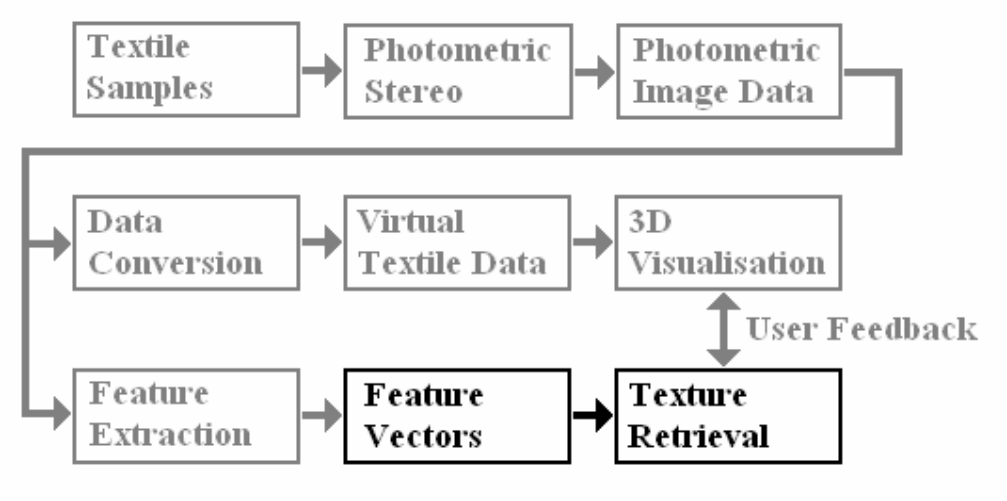


Figure 57: Stage two of the project data representation

The *texture retrieval* system requires that each *texture image* entry stored within the database should have a *feature vector*, a data structure capturing the fundamental properties of the image associated with it. Assigning a *feature vector* to each textile entry, allows the rapid comparison of *texture images* for similarity. For our experiments, we choose to implement ten different *rotation invariant* filter systems that operate in both the *spatial domain* and in the *frequency domain*. For the *spatial domain*, we choose to implement the *colour histogram* due to the simplicity of calculation. For the *frequency domain*, we choose to implement the *ring filter bank*, the *wedge filter bank*, the *Gabor filter bank* and the *Polarogram* due to their simplicity and sensitivity to periodicity in textile data. To investigate the ability of *frequency domain filter banks* to successfully perform *texture retrieval*, we choose to implement the *Schmid filter bank*, the *Leung-Malik filter bank*, the *MR4 filter bank* and the *MR8 filter bank* due to their *rotation invariant* natures.

Integrating the use of these *filter banks* with our *3D surface representation* is achieved using the following four stages:

(1) Converting each texture in the textile database into a *3D surface representation* and then into a *feature vector*
(2) Converting the target texture into a *3D surface representation* and then into a *feature vector*.

(3) Comparing the *feature vector* of the target image against all the pre-existing *feature vectors* of the textile database, and
(4) Sorting the results so that the best matches are at the top of the list.

In stages (1) and (2), the *3D surface representation* consists of the *albedo, gradient field* data in two axii (*partial surface derivatives*), and the *normalmap*. As noted in Chapter 2, surface derivatives are directional and therefore introduce artifacts that must be compensated for, before use in a *rotation invariant texture retrieval* system.

7.2 Organisation

The organisation of this chapter is as follows. We describe the criteria used for selecting the *texture retrieval* methods in section 7.3. Then we perform a detailed review of candidate methods in section 7.4. According to the criteria, we describe the ten methods for generating a *feature vector* in section 7.5, and present a quantitative assessment of *texture retrieval* methods in section 7.6, with the conclusions provided in section 7.7

7.3 Criteria

In our literature survey, we defined the following set of criteria that each candidate *texture retrieval* method must satisfy:

- *Must select visually similar textures using the surface representation*
  We would like the user to be able to select a target texture and have the retrieval system find those textures in the database that are as similar as possible to the target texture, particularly with regard to colour distribution and *micro-geometry* characteristics. As we are working with real-world textile images, It is of particular importance that the retrieval methods must be able to work with true-colour images (with at least eight bits for each of the red, green and blue colour channels).

- *Must be rotation invariant*

  Rotation invariance is the ability of a *texture retrieval* system to match two similar images regardless of their rotational orientation. Whenever more than one *photometric image* of a textile sample is made, each image will always have a unique rotation, due to slight differences in the position of each texture sample relative to the recording device. Consequently, this will distort any *feature vector* derived from this image. We describe *texture retrieval* methods that are able to overcome this problem as being *rotation invariant*. Of particular concern are the directional artifacts present in the surface normal fields.

- *Must have efficient memory usage*

  An uncompressed 512 x 512 pixel true-colour image with 16-bits of data per pixel for each of the red, green and blue colour channels will occupy 1.5 Mbytes of memory, while a image with 8-bits per colour channel will occupy 768 Kbytes of memory and a JPEG compressed image will still occupy more than 400 Kbytes of memory. Even with such high levels of data compression, these amounts of memory used are still far too high for *texture retrieval* to operate interactively. Consequently, any *feature vector* derived from a *texture image* must be considerably smaller than this. Given that the *texture database* may be located on a separate server, and accessible only via a dial-up connection, an upper limit of 8 Kbytes per *feature vector* is considered to be efficient memory usage.

As well as satisfying the three criteria specified above, the *texture retrieval* system must also be able to operate in *real-time*, implement distance functions that perform similarity matching between different database entries, and use the *texture retrieval* method that has the highest *accuracy rate*.

7.4 Using surface normal data

In Chapter 3 we identified how the *micro-geometry* of textile samples could be represented using the surface normal $(\mathbf{n})$, height data (h) and colour *albedo* (r,g,b). In Chapter 4 we described how *photometric stereo* could be used to acquire this data from real world textile samples. We also described how a collection of such images formed the Virtex photometric database.

Within this database, we use three images to represent each textile sample. These include the *albedo* image, the *gradient field* image and the *normalmap* image. The *albedo* image is simply a *photometric image* of the textile sample illuminated under optimal ambient lighting conditions such that there are no shadows. However, as mentioned previously, the normalmap contains directional artifacts and we therefore define the *gradient field* image by the following equation:

$$g(x, y) = (p(x, y))^2 + (q(x, y))^2 \qquad (7.4.1)$$

We perform this mathematical operation in order to eliminate the directional bias due to the use of a two dimensional coordinate system, and thus guarantee *rotation invariant* results.

However, one problem with images acquired using photometric methods is that the image acquision environment may vary between textile samples. Such conditions include the distance between the textile sample and the camera, the frame resolution, the focal length and shutter speed, and ambient colour temperature. Each of these will have a detrimental effect on the scale resolution, contrast and black-level settings of each *photometric image*. As described in section 3.3, we ensure that the digital camera is mounted at the same distance from each textile sample along with identical focal length. This avoids the problem of performing *texture classification* using materials that may have been acquired at different scales. We also operate the digital camera within a dark-room with the ceiling lighting switched off during the photometric acquisition process in order to avoid problems with contrast and black level illumination. The resulting

photometric images are also normalised to the precision of the framebuffer of the digital camera.

We describe this mathematical process in detail in the paper "Real-time per-pixel rendering of bump-mapped textures captured using *photometric stereo*" [Robb2004], and provide a detailed description in section 5.5.4. We present the complete set of textures in "Appendix B – The Texture Dataset"

7.5 The selected texture retrieval methods

In Chapter 2, we performed a review of the available *texture retrieval* methods. There are two hypotheses which we wish to investigate, the first being whether larger *feature vectors* offer any advantage over small *feature vectors* to *texture retrieval*, and the second being whether *feature vectors* composed of different types of *filter bank* offers any advantage over *feature vectors* composed from a single type of *filter bank*.

We can categorize textile samples into four basic groups:

- Textures with no pattern due to fine weaving, generating no dominant signals after transformation in the *frequency domain*.
- Textures with a striped pattern, generating a single dominant signal in the *frequency domain*
- Textures with a regular square or hexagonal patterns, generating two or three dominant signals at similar frequencies but different directions in the *frequency domain*
- Textures with a rectangular or trapezoid block pattern, generating two or more different dominant signals at different frequencies and different directions in the *frequency domain*

For the first class of textures, textures with no dominant signal, we choose to investigate the use *of colour histograms* as described by Swain and Funt as these are naturally

rotation invariant and do not depend upon transformation into the *frequency domain* [Swain1991] [Funt1991]. We also choose to investigate the *Schmid filter bank* due to the *natural rotation invariant* ability of each filter. This *filter bank* is of particular use when the texture detail is so fine as to only appear as dots or spots.

For the second class of textures, textures with a single dominant signal in the *frequency domain*, we choose to investigate the *ring filter bank* [Randen1999] and the *Polarogram* [Davis1981] [Wu2003]. Each of these *filter banks* generates a *feature vector* that is insensitive to direction. These *filter banks* are particularly useful when the texture detail appears as lines or stripes.

For the third class of textures, textures with two or more dominant signals at similar frequencies but different directions in the *frequency domain*, we choose to investigate the use of the *wedge filter bank*. The *wedge filter bank* is insensitive to frequency, but is sensitive to direction. This filters bank is particularly useful when there texture detail appears as regular spot or dots patterns.

For the fourth class of texture, textures with two or three dominant signals with different frequencies and different directions in the *frequency domain*, we choose to investigate the *Gabor filter bank* [Bovik1990] [Jain1990] [Randen1999], the *Leung-Malik filter bank* [Leung2001], the *MR4 filter bank* and the *MR8 filter bank*. The *Gabor filter bank* has the advantage of having filters which are sensitive to both direction and frequency. The *Leung-Malik filter bank* has a combination of filters which are both *rotation invariant* and sensitive to direction. We select the *MR4 filter bank*, and the *MR8 filter bank* [Varma2005] due to their use of "collapsing" inputs to reduce the size of the *feature vector* generated from identical inputs. These *filter bank* will be most successful at selecting between textures when there is only one dominant direction.

7.5.1 Summary

Our ten selected methods are thus as follows:

Colour Histograms: This method generates a *feature vector* based on the frequency of different pixel intensities. *Colour histograms* are *rotation invariant* as they are first order statistics. The *feature vector* consists of seven hundred and sixty eight floating-point values.

Ring filters: This method operates on second order statistics and generates a *feature vector* based on frequency only. The *ring filter bank* is *rotation invariant*. The *feature vector* consists of sixty floating-point values.

Wedge filters: This method also operates on the power spectrum, but generates a *feature vector* based on direction only. The *wedge filter bank* is directionally sensitive, and is not *rotation invariant* without additional post-processing. The *feature vector* consists of sixty floating-point values.

Gabor filters: This method also operates on the power spectrum, and generates a *feature vector* based on a selected combination of frequencies and directions. By default, this method is directionally sensitive, but is *rotation invariant* with the use of post-processing. The *feature vector* consists of ninety floating-point values.

Schmid filter bank: This method operates on the power spectrum, and is *rotation invariant*. Each *Schmid filter* is sensitive to a complex sinusoidal wave pattern without any directional sensitivity. The *feature vector* consists of thirty-nine floating-point values.

Leung-Malik filter bank: This method operates on the power spectrum and is directionally sensitive. It consists of a large collection of different types of filter, including three sets of edge filter; three sets of *bar filter*, *Gaussian filters* and *Laplacian-of-Gaussian filters*. By default, this method is directionally sensitive, but is *rotation invariant* with the use of post-processing. The *feature vector* consists of one hundred and forty-four floating-point values.

Maximum Response 8 filter bank: Similar to the *Leung-Malik filter bank*, but with only the maximum response of each *edge filter* and *bar filter* being stored for each of the three frequencies. As with the *Leung-Malik filter bank*, this *MR-8 filter bank* operates in the power spectrum and is *rotation invariant* due to the selection of the maximum response. The *feature vector* consists of twenty-four floating-point values.

Maximum Response 4 filter bank: Similar to the *Leung-Malik filter bank*, but only the maximum response of each of the edge and bar filters is stored for only one frequency. As with the *Leung-Malik filter bank* and the *MR-8 filter bank*, the *MR-4 filter bank* operates in the power spectrum, and is *rotation invariant* due to the selection of the maximum response. The *feature vector* consists of twelve floating-point values.

Polarogram This method generates a *feature vector* based on the sum of energy levels for each direction in the *frequency domain*. The *feature vector* consists of three hundred and sixty floating-point values.

Combined filter banks: All of the above methods combined. This operates in both the power-spectrum and with first order statistics. Without post-processing this stage would be directionally sensitive. *The feature vector* consists of one thousand and fifty three floating-point values.

We provide additional details on the final list of ten candidate methods in Table 8 and provide further details of each method in the next section. In the following table, "Albedo/Surface Texture" refers to the ability of the *texture retrieval* method to work with both the *albedo* image and *gradient field* data. "*Rotation invariant*" indicates whether the *texture retrieval* method consists of filters which are all *rotation invariant*. "Dimensionality" indicates the relative size of the *feature vector* generated from the *filter bank*. A *texture retrieval* method with a small number of filters is said to have low dimensionality, while a *texture retrieval* method with a large number of filters is said to have high dimensionality. The final column, "Fourier spectrum", indicates whether the *texture retrieval* method requires transformation into the *frequency domain* or not.

Method	**Albedo/ Surface Texture**	**Rotation invariant**	**Dimensionality (Low <15) (High >=15)**	**Fourier Spectrum**
Colour Histograms	Both	Yes	High	No
Ring filter bank	Both	Yes	High	Yes
Wedge filter bank	Both	No	High	Yes
Gabor filter bank	Both	No	High	Yes
Schmid filter bank	Both	Yes	Low	Yes
Leung-Malik filter bank	Both	No	High	Yes
MR-4 filter bank	Both	Yes	Low	Yes
MR-8 filter bank	Both	Yes	Low	Yes
Polarogram	Both	Yes	High	Yes
Combined filter banks	Both	Yes	High	Yes

Table 8: Summary of the selected texture retrieval methods

We also choose to use the Euclidean distance calculation to determine similarity matching between texture samples for each *texture retrieval* method as this is simple to calculate and common in the literature.

One of the concerns that we have with the use of large *filter banks* with high dimensionality, is the hazard of having duplication or redundancy of information within each *feature vector*. One possible solution to this problem is the use of Principal Component Analysis (PCA). In this method, the images generated from the *3D surface representation* are stacked together and the covariance matrix calculated for each pair of axii, then calculating the eigenvectors and eigenvalues from the covariance matrix. The resulting set of eigenvectors and eigenvalues are then used to form a *feature vector*

which can be used to perform *texture retrieval*. The advantage of this method is that the use of PCA greatly improves the *accuracy rate* of *texture retrieval*. The disadvantage of this method is that the size of each *feature vector* generated using PCA is dependent entirely upon the dimensions of the sample images. However, the data is positionally sensitive and as texture are often characterized by their higher frequency information, but if necessary, those principal components that do not contribute much variability to the data can be discarded. We choose not to investigate the potential of PCA for *texture retrieval*.

7.6 Implementation of the selected texture retrieval methods

In section 7.4, we selected ten methods, each of which use a set of *texture images* as input in order to generate *feature vectors* which can then be used for *texture retrieval*. In this section, we propose a data representation that is used to implement and compare these *texture retrieval* methods. This section provides a summary of the common properties of the ten *texture retrieval* methods. We begin this chapter by providing an overview of the use of the FFT and IFFT in order to implement *filter banks*.

7.6.1 Filter banks and the Fast Fourier Transform (FFT)

One of the earliest known publications in the field of signal processing is the paper written by Fourier in 1807 [Fourier1822]. In his paper, Fourier describes that a function "having a spatial period λ, can be synthesized by a sum of harmonic functions whose wavelengths are integral submultiples of λ". Fourier analysis is a method of breaking down a complex wave into a set of fundamental sinusoidal waves, each with a unique frequency and amplitude. The resulting series of terms is mathematically using the 2D FFT (7.6.1.1) and IFFT formulas (7.6.1.2). Research into the applications of Fourier series continued for well over 130 years [Littlewood1937].

$$F(u,v) = \frac{1}{rs}\sum_{x=0}^{r}\sum_{y=0}^{s} f(x,y)e^{-j2\pi(u\frac{x}{r}+v\frac{y}{s})} \quad (7.6.1.1)$$

Where: $F(u,v)$ is the signal in the *frequency domain*,

$f(x,y)$ is the signal in the *spatial domain*,

(u,v) are coordinates in the *frequency domain* in cycles per image width and cycles per image height respectively.

(x,y) are coordinates in the *spatial domain*,

and (r,s) are the dimensions of the domain

$$f(x,y) = \sum_{u=0}^{r}\sum_{v=0}^{s} F(u,v)e^{j2\pi(u\frac{x}{r}+v\frac{y}{s})} \qquad (7.6.1.2)$$

Where: $F(u,v)$ is the signal in the *frequency domain*,

$f(x,y)$ is the signal in the *spatial domain*,

(u,v) are coordinates in the *frequency domain* in cycles per image width and cycles per image height respectively.

(x,y) are coordinates in the *spatial domain*,

and (r,s) are the dimensions of the domain

When applied to *texture classification*, the FFT is of particular use. By using the FFT to calculate the output responses of each filter in a *texton filter bank*, it becomes possible to measure the similarity between two images while at the same time, including a tolerance for varying position and scale.

For the purposes of this thesis, we chose to implement the software used to evaluate the *rotation invariant texture retrieval* stage of our thesis using command line software written using C++ and running on a Linux system. There were several reasons for doing this. The first reason was that the use of shell scripts allowed the automated batch processing of large number of textile sample images using a multi-processor system. This allowed the generation of *feature vectors* to be performed without supervision, which proved to be a time-consuming task due to the use of the FFT with large images. At the time that this research was conducted, no GPGPU (General Purpose GPU) software such as CUDA or OpenCL was available). The use of command line programs allowed for small shell scripts to be rapidly constructed in order to select *particular filter bank* sets to use.

Before we provide a more detailed specification of each of these *filter banks*, we describe the process in which the *feature vectors* for the *wedge filter bank*, *ring filter bank*, *Gabor filter bank*, *Schmid filter bank*, *Leung-Malik filter bank*, *MR-4 filter bank* and *MR-8 filter banks* are calculated for each *texture image*. We use the FFT to transform the target *texture image* into the *frequency domain*, and then combine this image with the *frequency domain* image of each filter in the selected *filter bank*. We

then apply the IFFT to convert the resulting image back into the *spatial domain*, and calculate the response of that filter by calculating the sum of squares of all the pixel values in the resulting image. The resulting set of responses form the *feature vector* for that particular *filter bank*. We also perform an additional processing stage for those *feature vectors* derived from those *filter banks* that are not fundamentally *rotation invariant*. These include the *Wedge filter bank*, the *Gabor filter bank*, the *Leung-Malik filter bank*, the *MR-4 filter bank* and the *MR-8 filter bank*. This stage involves implementing the circular shift method described by Zhang [Zhang2002]. Using this method, we compare pairs of *feature vectors* together by shifting through every possible pair of orientations, evaluating the level of similarity and returning the highest value found. Having described the basic operation of the *filter bank* and the FFT, we now describe each individual *filter bank* in detail. These are as follows:

- The *ring filter bank*
- The *wedge filter bank*
- The *Gabor filter bank*
- The *Schmid filter bank*
- The *Leung-Malik filter bank*
- The *MR4 filter bank*
- The *MR8 filter bank*
- The *Polarogram*

We also consider *colour histograms* and the combined use of all *filter banks*. In total, we consider ten different *texture retrieval* methods. We begin by describing the *spatial domain* methods in detail (the *Colour Histogram*) followed by the *frequency domain* methods.

7.6.2 The Histogram and Colour Histogram

Histograms and Colour histograms are one of the simplest ways to generate a *feature vector* of an image. The *histogram* for a single image is defined by three parameters; the minimum and maximum values of the range, and the number of separate bins within

that range. For every pixel within the image found to belong within the range of a particular bin, the value of that bin is incremented. The minimum number of bins that any *histogram* can contain is one, with no limit on the maximum number of bins unless the data being analysed consists of discrete logical representations such as integers. In this case, the precision of the data type defines the upper limit. For monochrome or single channel images, the *histogram* is constructed from the gray scale values. For colour or red/green/blue images, the *histogram* can be constructed in two ways. The first way is to construct separate *histograms* for each colour channel. Alternatively, a three-dimensional *histogram* or *colour histogram* can be constructed by splitting the three-dimensional colour space (the colour cube) into separate sub-cubes and assigning a *histogram* bin to each sub-cube. Due to its ease of calculation, the *colour histogram* is a popular choice as a *feature vector* for *texture retrieval*. Determining the level of similarity between two images, simply involves calculating the sum of differences squared between the matching bins of each *colour histogram*, saving the results into a temporary array, sorting the array in descending order, and returning the required number of entries from the top of the list. For this thesis, we use a *colour histogram* of 512 bins as specified by Swain and Ballard [Swain1991]. This number was derived from experimentation with a *texture database* consisting of sixty-six pictures of consumer products.

7.6.3 The ring filter bank

Ring filters are one of the basic types of *filter bank* used in combination with the FFT and IFFT [Randen1999], with each *ring filter bank* comprised of a number of *ring filters,* with each filter having a unique response frequency. Because each *ring filter* has no directional sensitivity, the *filter bank* as a whole, also has no directional sensitivity and is therefore *rotation invariant*. Each *ring filter* has the effect of summing together all the signal energy for all directions in a selected frequency range. We combine the target frequency of the *ring filter* with a Gaussian equation to give smoothly blended edges in order to prevent any *spatial domain* "ringing" that sharp edges on a filter might generate. We present the equation of the *ring filter* in (7.5.3.1), along with individual images of *ring filters* in Figure 58, and the entire set in (Figure 59).

$$F_r(r,\sigma_r,r_{centre}) = e^{(-\frac{(r-r_{centre})^2}{2\sigma_r^2})} \qquad (7.6.3.1)$$

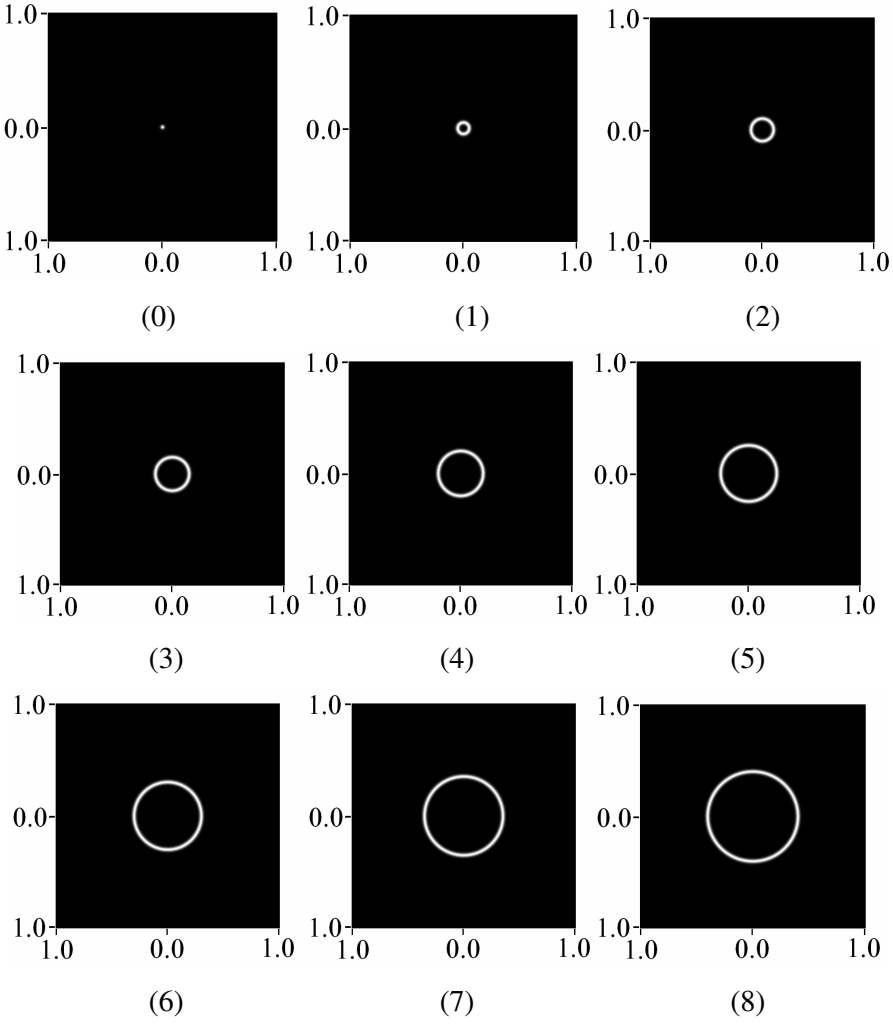
Where: $F_r(r,\sigma_r,r_{centre})$ is the Ring filter function,

r is the radius of the ring in cycles/image width, given by

$$r = \sqrt{u^2 + v^2}$$

σ_r is the standard deviation of the ring in cycles/image width

and r_{centre} is the centre point of the ring in cycles/image width



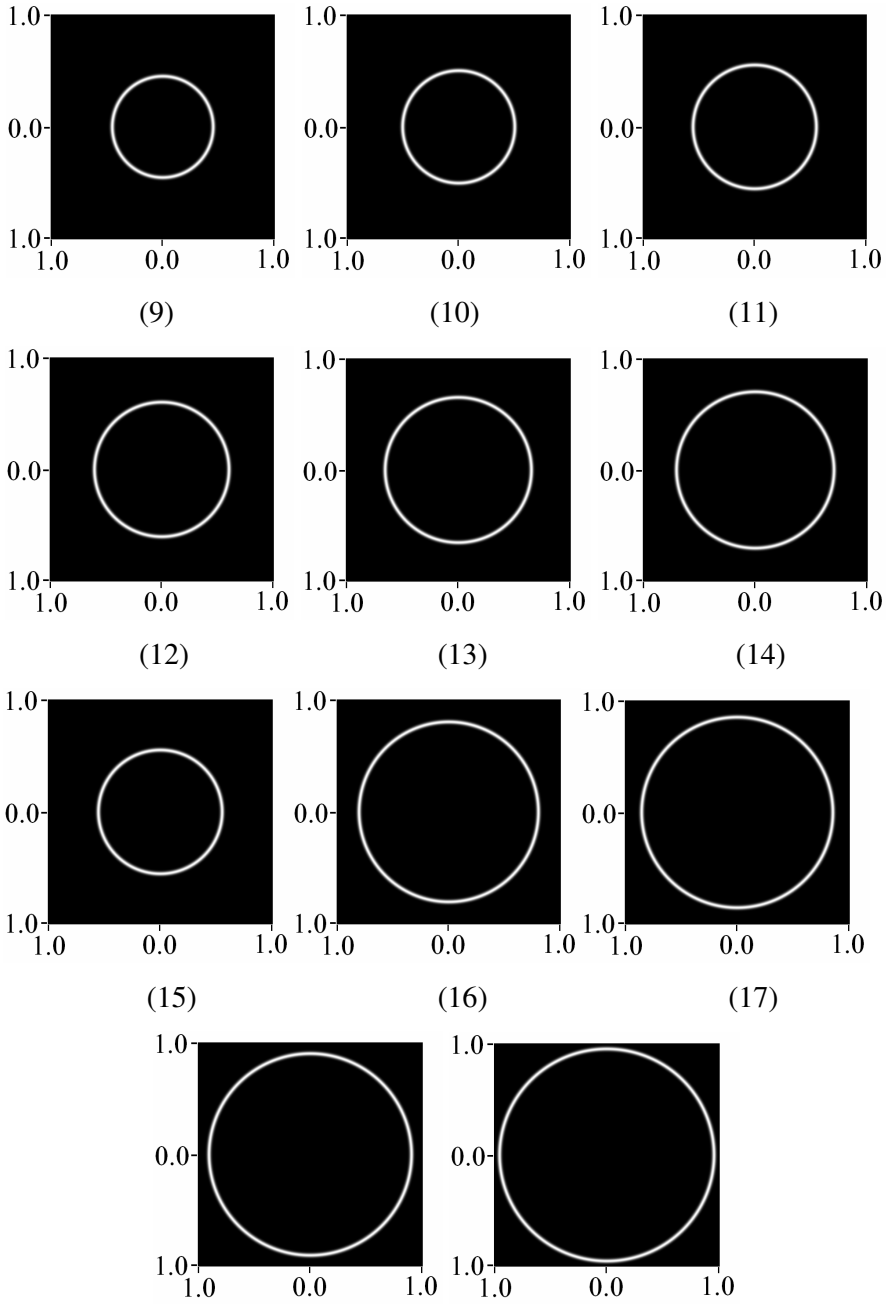


Figure 58: Ring filters in the frequency domain

(On each axis, units are shown as fraction of the Nyquist frequency).

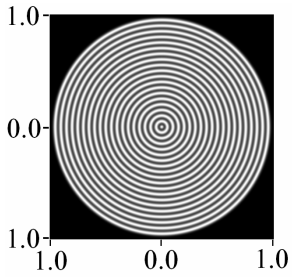


Figure 59: All Ring Filters in the frequency domain
(On each axis, units are shown as fraction of the Nyquist frequency).

For all of the experiments conducted by this thesis, we choose a *ring filter* system with twenty filters, with an output for each colour channel. This was determined from analysis of the transformation of the *texture images* into the *frequency domain*, where it was observed that the peak signal responses could be closely modeled by a Gaussian curve with a standard deviation of 0.0125. Thus, the peak response Nyquist frequency of each *ring filter* ranges from 0.0 to 1.0 in increments of 0.05, with a standard deviation of 0.0125. This results in *a feature vector* size of sixty floating-point values.

We calculate the result of each individual floating-point output for every filter system using the method now described. Given the image of the texture in the *spatial domain* $I_t(x, y)$, and the image of the filter in the *spatial domain* $I_f(x, y)$, then we calculate each filter output value O_n as follows:

Transform the filter image $I_f(x, y)$ and *texture image* $I_t(x, y)$ from the *spatial domain* into the *frequency domain* to obtain the *frequency domain* images I'_f and I'_t:

$$\begin{aligned} I'_t(u,v) &= F(I_t(x,y)) \\ I'_f(u,v) &= F(I_f(x,y)) \end{aligned} \qquad (7.6.3.2)$$

Combining the two together in the *frequency domain* to give $r(x, y)$:

$$r(u,v) = I'_t(u,v) \cdot I'_f(u,v) \qquad (7.6.3.3)$$

Convert the image back into the *spatial domain* using the inverse FFT to give $e(x, y)$:

$$e(x, y) = f(r(x, y)) \qquad (7.6.3.4)$$

Calculating the filter output value:

$$O_n = \sum_{x=0}^{x<=r} \sum_{y=0}^{y<=s} e(x, y)^2 \qquad (7.6.3.5)$$

We perform this calculation for every filter in every *filter bank*, with the exception of the *histogram* method.

7.6.4 The wedge filter bank

Wedge filters are another basic type of *filter bank* used in combination with the discrete FFT and IFFT [Coggins1982] [Randen1999], with a *wedge filter bank* being composed of a number of *wedge filters* with different directional sensitivity. Since each *wedge filter* is directionally sensitive, the *filter bank* is also directionally sensitive, and thus is not *rotation invariant*. Each *wedge filter* has the effect of summing together all the signal energy for all frequencies in a selected direction. We combine the target frequency of the *wedge filter* with a Gaussian equation to give smoothly blended edges in order to prevent any distortion of the signal that sharp edges on a filter might generate. We present the equation of the *wedge filter* in (7.5.4.1), with individual image of each *wedge filter* in the *frequency domain* in (Figure 60), and combined in (Figure 61).

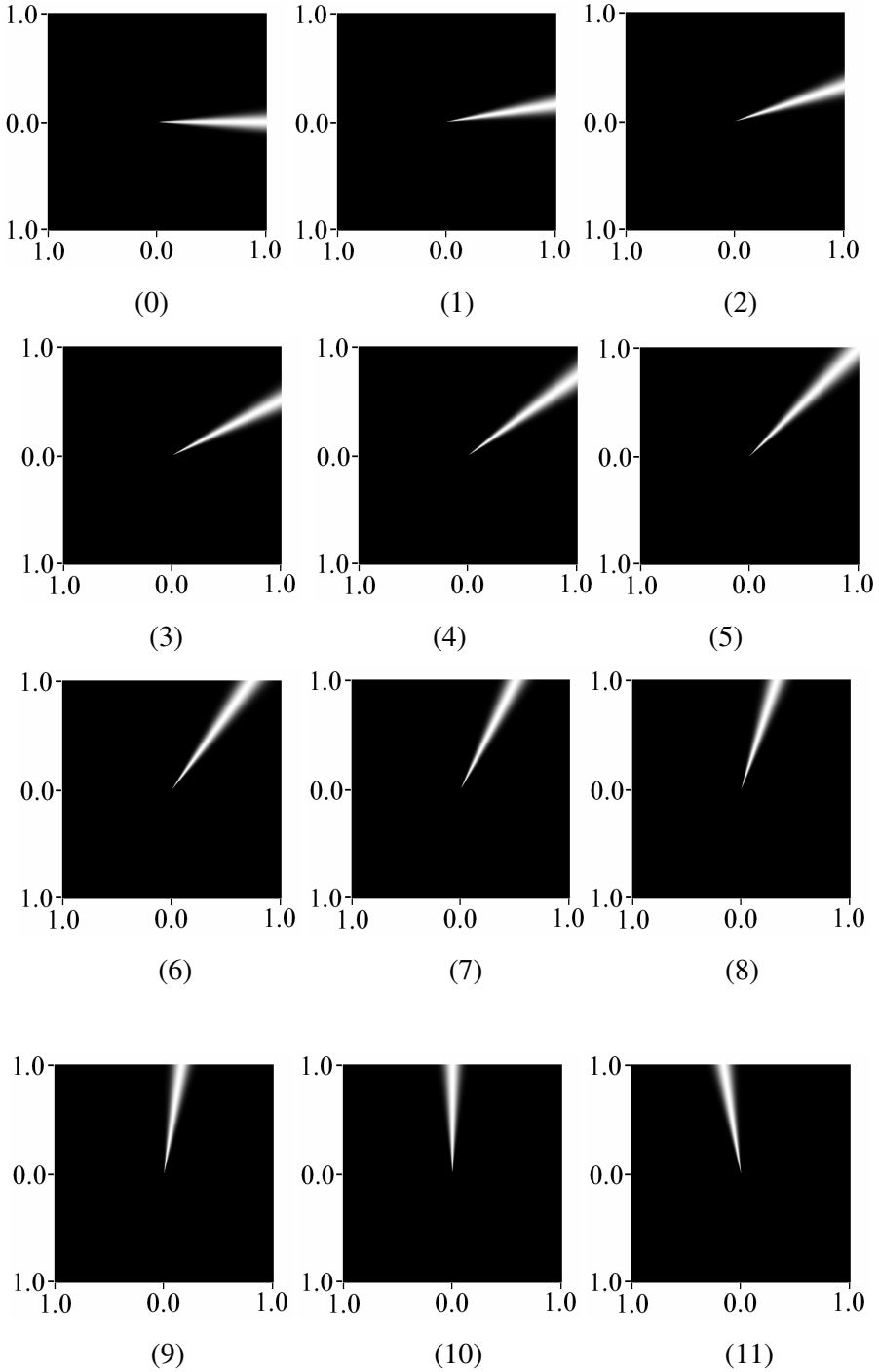
$$F_w(\theta, \sigma_\theta, \theta_{centre}) = e^{(-\frac{(\theta-\theta_{centre})^2}{2\sigma_\theta^2})} \qquad (7.6.4.1)$$

Where: $F_w(\theta, \sigma_\theta, \theta_{centre})$ is the *wedge filter* function,

θ is the angle, given by $\theta = \tan^{-1}(u, v)$,

σ_θ is the standard deviation in cycles/image width,

and θ_{centre} is the main direction of the *wedge filter*.



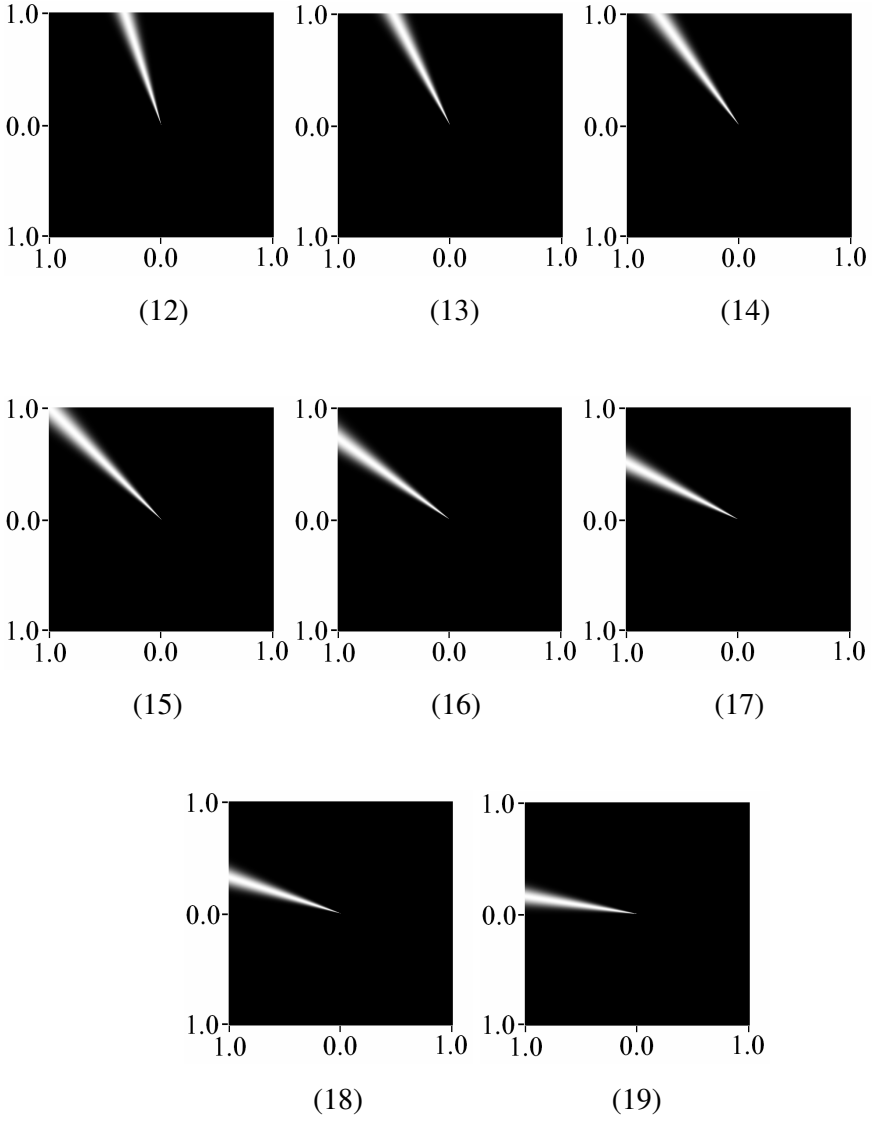


Figure 60: Wedge filters in the frequency domain

(On each axis, units are shown as fraction of the Nyquist frequency).

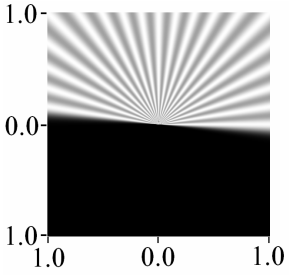


Figure 61: All wedge filters in the frequency domain

(On each axis, units are shown as fraction of the Nyquist frequency.
Also, as only the real component of the FFT is used, we assume the power spectrum to be symmetrical, and so only sample half of the frequency space)

For all of the experiments conducted by this thesis, we chose a *wedge filter* system with twenty filters, with an output for each colour channel. This was determined from analysis of the transformation of the *texture images* into the *frequency domain*, where it was observed that the peak signal responses could be closely modeled by a Gaussian curve with a standard deviation of 0.05. Thus, the peak response angle for each *wedge filter* ranges from 0.0 to 160 degrees in increments of 9 degrees with a standard deviation of 0.05. This results in *a feature vector* size of sixty floating-point values.

7.6.5 The Gabor filter bank

Gabor filter banks are an extension of both *ring filter banks* and *wedge filter banks*, in the sense that each *Gabor filter* has sensitivity to both a particular frequency and a directional angle. In the *spatial domain* the Gabor filter is a cosine wavelet modulated by a Gaussian envelope. Because *Gabor filters* inherit directional sensitivity from the *wedge filter* component, they are not rotation-invariant. Consequently, the *Gabor filter bank* is also not *rotation invariant*. To generate the Gabor filter in *the frequency domain*, we transform the *spatial domain* image into the *frequency domain* using a discrete FFT. We present the equation of the *Gabor filter* in (5.5.4.1), with individual images of *Gabor filters* in the *frequency domain* presented in (Figure 62), and combined together in (Figure 63).

$$F_g(r,\sigma_r,r_{centre},\theta,\sigma_\theta,\theta_{centre}) = \exp(\frac{-(r-r_{centre})^2}{2\sigma_r^{\,2}}).\exp(\frac{-(\theta-\theta_{centre})^2}{2\sigma_\theta^{\,2}}) \qquad (7.6.5.1)$$

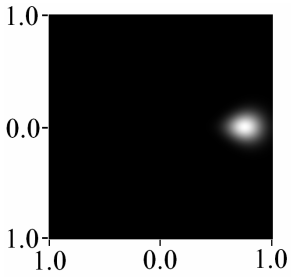
Where: $F_g(r,\sigma_r,r_{centre},\theta,\sigma_\theta,\theta_{centre})$ is the Gabor filter function,

r_{centre} is the median radius for the ring component in cycles/image width,

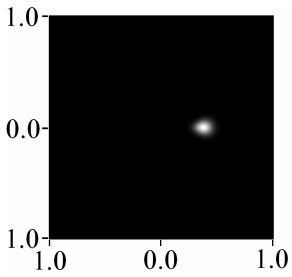
θ_{centre} is the median angle for the wedge component,

σ_r is the standard deviation for the radius in cycles/image width,

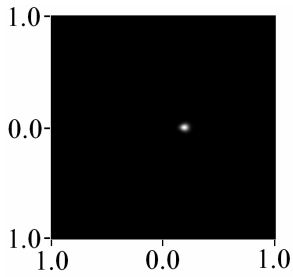
and σ_θ is the standard deviation for the angle.



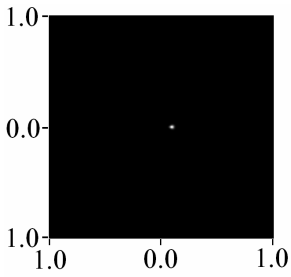
(0,0)



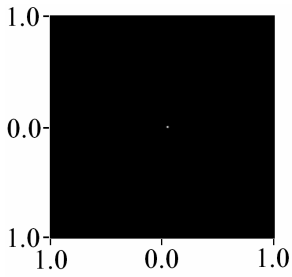
(0,1)



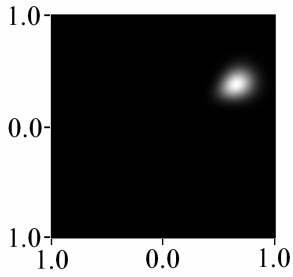
(0,2)



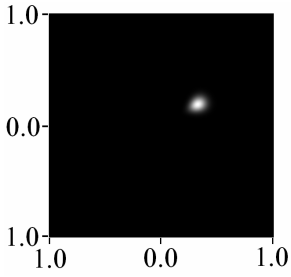
(0,3)



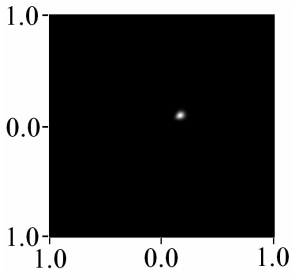
(0,4)



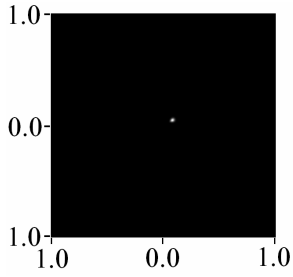
(1,0)



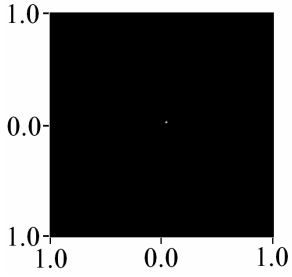
(1,1)



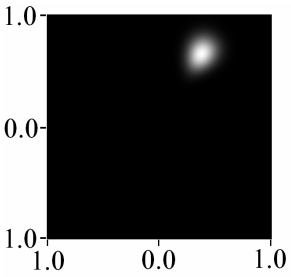
(1,2)



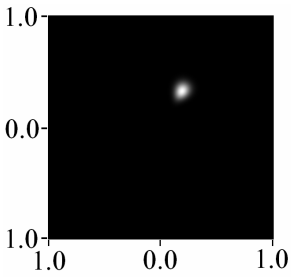
(1,3)



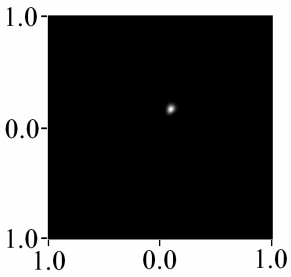
(1,4)



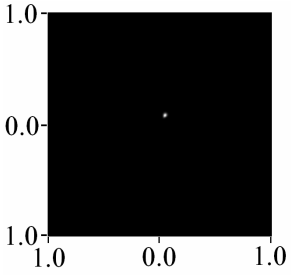
(2,0)



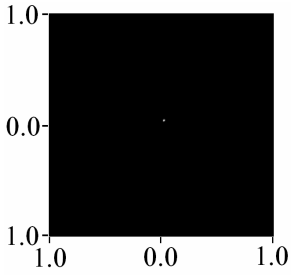
(2,1)



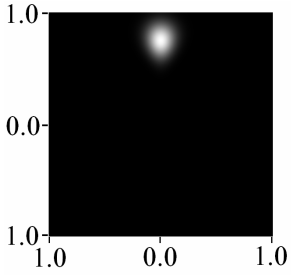
(2,2)



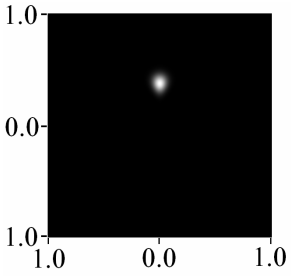
(2,3)



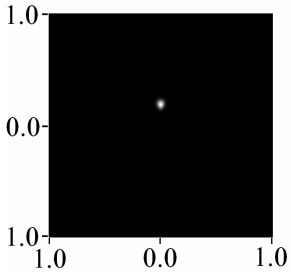
(2,4)



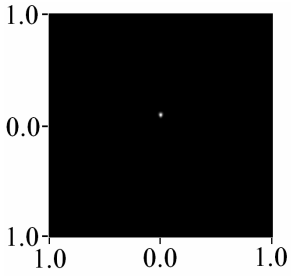
(3,0)



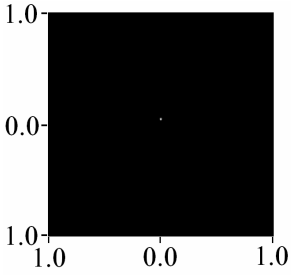
(3,1)



(3,2)



(3,3)



(3,4)

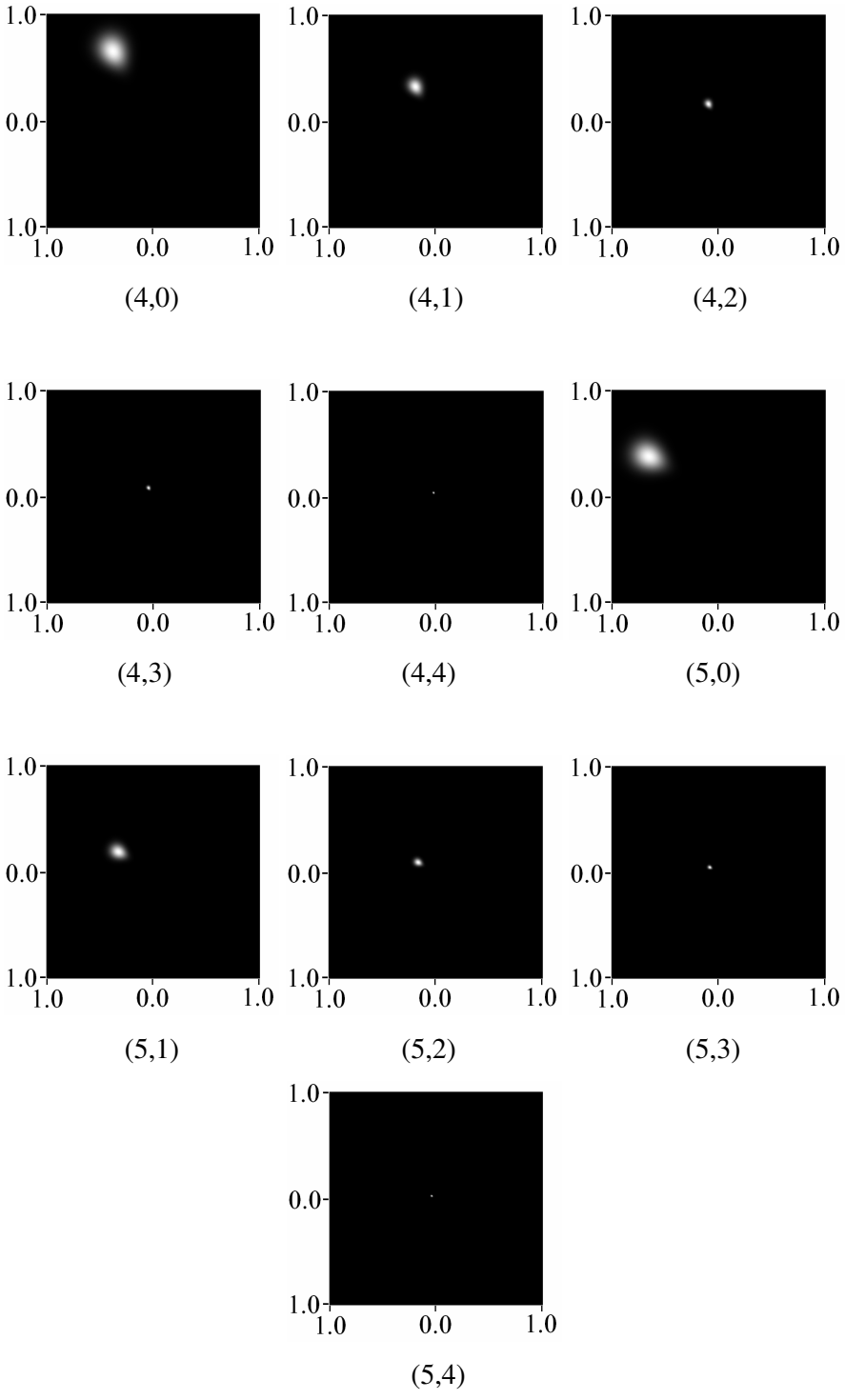


Figure 62: Gabor filters in the frequency domain

(On each axis, units are shown as fraction of the Nyquist frequency)

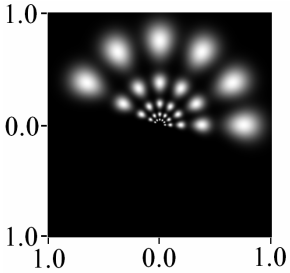


Figure 63: All Gabor filters in the frequency domain
(On each axis, units are shown as fraction of the Nyquist frequency.
Also, as only the real component of the FFT is used, we assume the power spectrum to be symmetrical, and so only sample half of the frequency space)

For this thesis, we choose the *Gabor filter bank* specified by Jain [Jain1991]. This *filter bank* consists of five frequency banks combined with six filter directions. This provides a *filter bank* consisting of thirty different filters, with an output for each colour channel. This results in *a feature vector* size of ninety floating-point values. We define the parameters of these filter bands in Table 9 and Table 10, with the constant C_r being defined as $2\sqrt{2\ln(2)}$ or 2.35482005

Frequency band	Frequency (fraction of the Nyquist frequency)	Standard Deviation (fraction of the Nyquist frequency)
1	$\frac{3}{4}$	$\frac{C_r}{4}$
2	$\frac{3}{8}$	$\frac{C_r}{8}$
3	$\frac{3}{16}$	$\frac{C_r}{16}$
4	$\frac{3}{32}$	$\frac{C_r}{32}$
5	$\frac{3}{64}$	$\frac{C_r}{64}$

Table 9: Table of Gabor filter frequency bands

Angular band	Angle (degrees)	Standard Deviation (degrees)
1	0	$\frac{30}{2C_r}$
2	30	$\frac{30}{2C_r}$
3	60	$\frac{30}{2C_r}$
4	90	$\frac{30}{2C_r}$
5	120	$\frac{30}{2C_r}$
6	150	$\frac{30}{2C_r}$

Table 10: Table of Gabor filter bank angular bands

7.6.6 The Schmid filter bank

Schmid filters [Schmid2001] are a modified version of the *ring filter*. While a *ring filter* is sensitive only to a particular frequency in all directions, a *Schmid filter* is defined by the convolution of a sinusoidal wave function modulated by a Gaussian envelope. We present the general equation of the *Schmid filter* in (7.6.6.1). The *Schmid filter bank*

consists of thirteen *rotation invariant* filters with the (σ,τ) pair is assigned the values (2,1), (4,1), (4,2), (6,1), (6,2), (6,3), (8,1), (8,2), (8,3), (10,1), (10,2), (10,3) and (10,4), and where $F_0(\sigma,\tau)$ is added to obtain a zero *DC component*. As each individual filter is *rotation invariant*, the complete *filter bank* is also *rotation invariant*. We present the complete *Schmid filter bank* in (Figure 64).

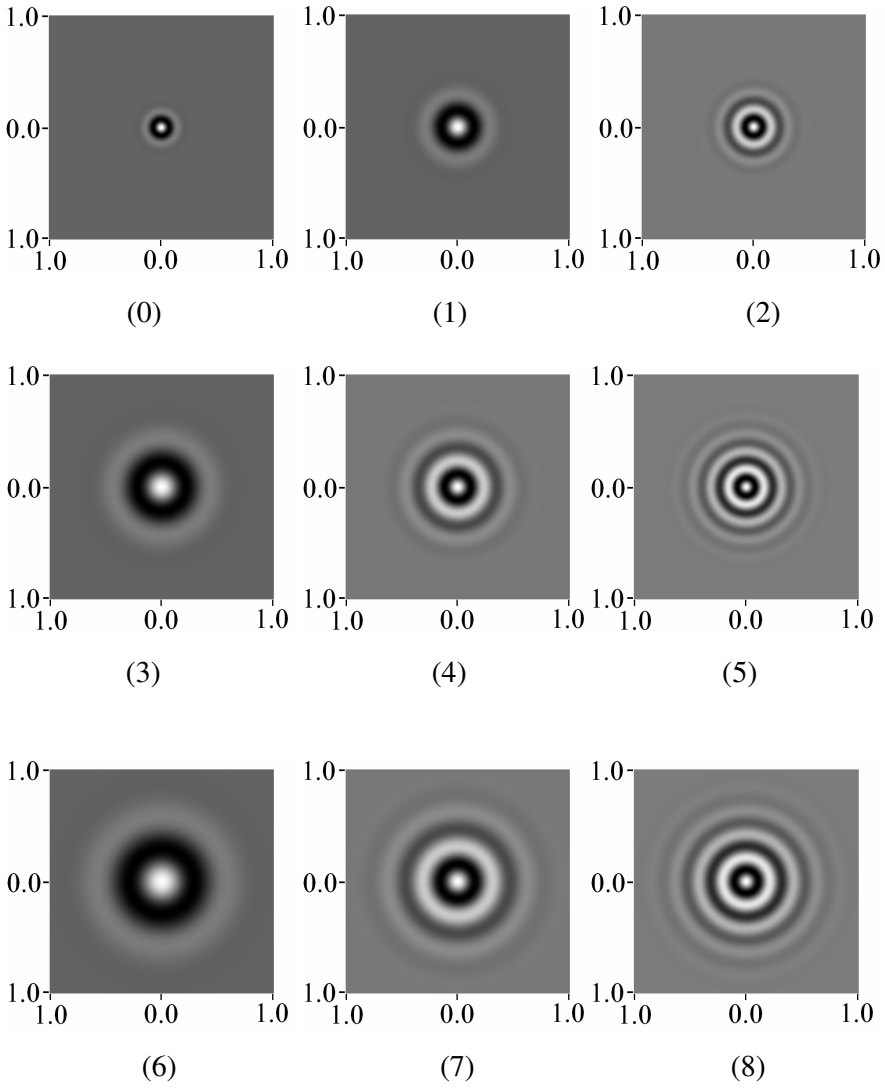
$$F_s(r,\sigma,\tau) = F_0(\sigma,\tau) + \cos\left(\frac{\pi.r.\tau}{\sigma}\right) e^{-\frac{r^2}{2\sigma^2}} \qquad (7.6.6.1)$$

Where: $F_0(\sigma,\tau)$ is added to obtain a zero *DC component*,

σ is the standard deviation in cycles/image width,

and τ is the number of cycles of the harmonic function,

within the Gaussian envelope of the filter.



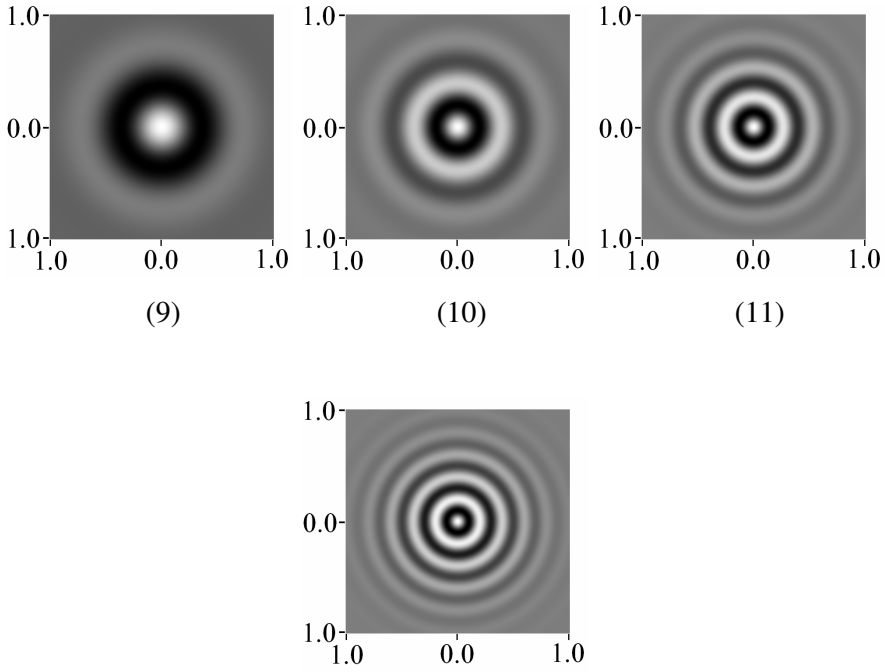


Figure 64: The Schmid filter bank in the frequency domain
(On each axis, units are shown as fraction of the Nyquist frequency)

For this thesis, we choose to use the standard set of thirteen filters as determined by Schmid [Schmid2001]. Schmid derived the set of filters from *texton analysis* using k-means clustering. In this *filter bank*, Schmid chose to use thirteen filters with σ ranging between 2.0 and 10.0, and τ ranging between 1.0 and 4.0, while avoiding large values of τ at small scales. This results in *a feature vector* size of thirty-nine floating-point values.

7.6.7 The Leung-Malik filter bank

The *Leung-Malik filter bank* is a set of forty-eight filters comprised from four different types of basic filter. The *filter bank* includes eighteen edge filters (first derivative of the *Gaussian filter*) at six different orientations and three different frequencies; eighteen bar filters (second derivative of the *Gaussian filter*) at six different orientations and three frequencies, four *rotation invariant Gaussian filters*, and eight *rotation invariant*

Laplacian-of-Gaussian filters [Marr1980]. We present the equations of these filters in (6.5.6.1), (6.5.6.2), (6.5.6.3), (6.5.6.4), and the frequency space images in (Figure 65), (Figure 67), (Figure 68) and (Figure 66).

$$F_{gm}(\sigma, r) = \frac{1}{2\pi\sigma_r^{\,2}} e^{-\frac{r^2}{2\sigma_r^{\,2}}} \qquad (7.6.7.1)$$

Where: $F_{gm}(x, y)$ is the Gaussian filter function

and σ_r is the standard deviation of the radius in cycles/image width

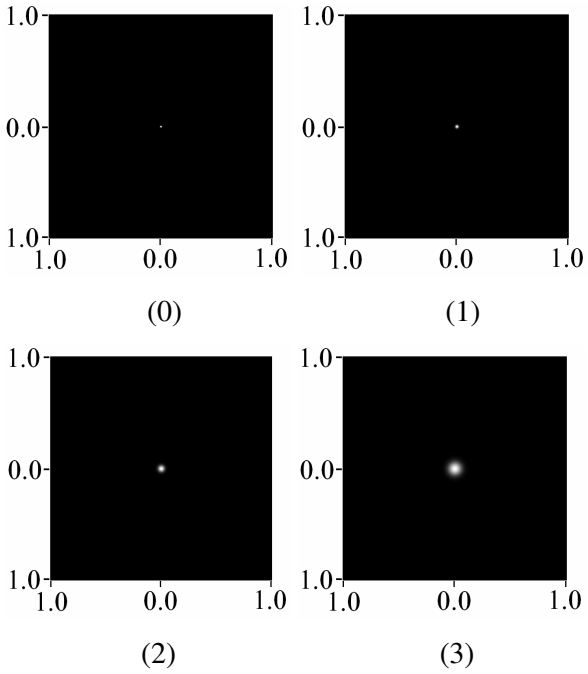


Figure 65: Gaussian filters in the frequency domain

(On each axis, units are shown as fraction of the Nyquist frequency)

$$F_{LoG}(\sigma, r) = -\frac{1}{\pi\sigma^4} e^{-\frac{r^2}{2\sigma^2}} \left[1 - \frac{r^2}{2\sigma^2} \right] \qquad (7.6.7.2)$$

Where: $F_{LoG}(\sigma, r)$ is the *Laplacian-of-Gaussian* filter function

and σ is the standard deviation of the radius in cycles/image width

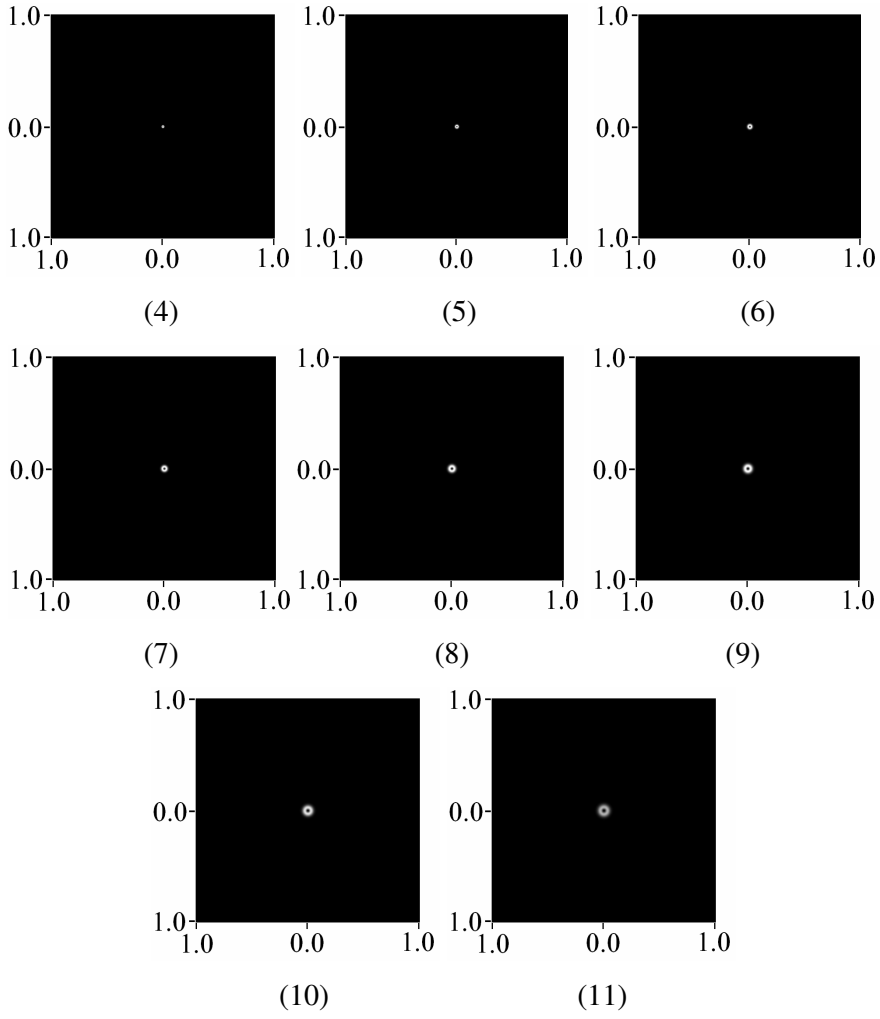


Figure 66: Laplacian-of-Gaussian filters

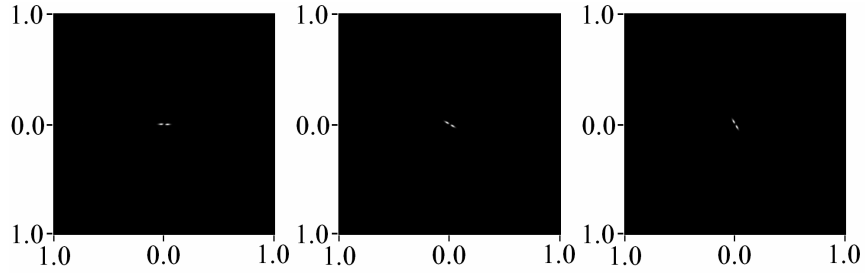
(On each axis, units are shown as fraction of the Nyquist frequency)

$$G'(u,v,\theta) = \frac{u.\cos(\theta) + v.\sin(\theta)}{2\pi\sigma_r^{\,4}} e^{-\frac{r^2}{2\sigma_r^2}} \qquad (7.6.7.3)$$

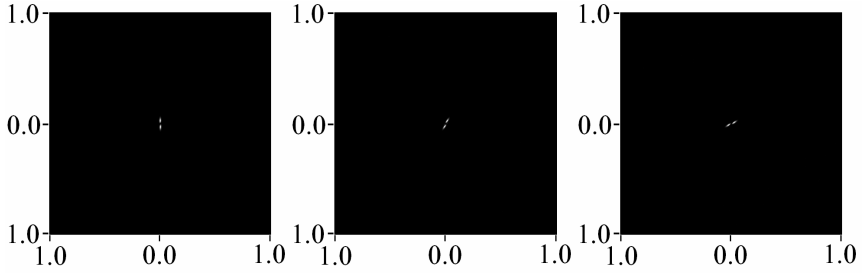
Where: $G'(u,v,\theta)$ is the Gaussian first derivative filter,

θ is the angle, given by $\theta = \tan^{-1}(u,v)$

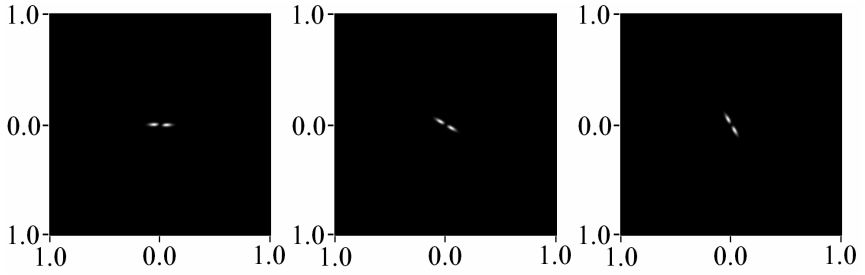
and σ_r is the standard deviation of the radius.



(12) (13) (14)



(15) (16) (17)



(18) (19) (20)

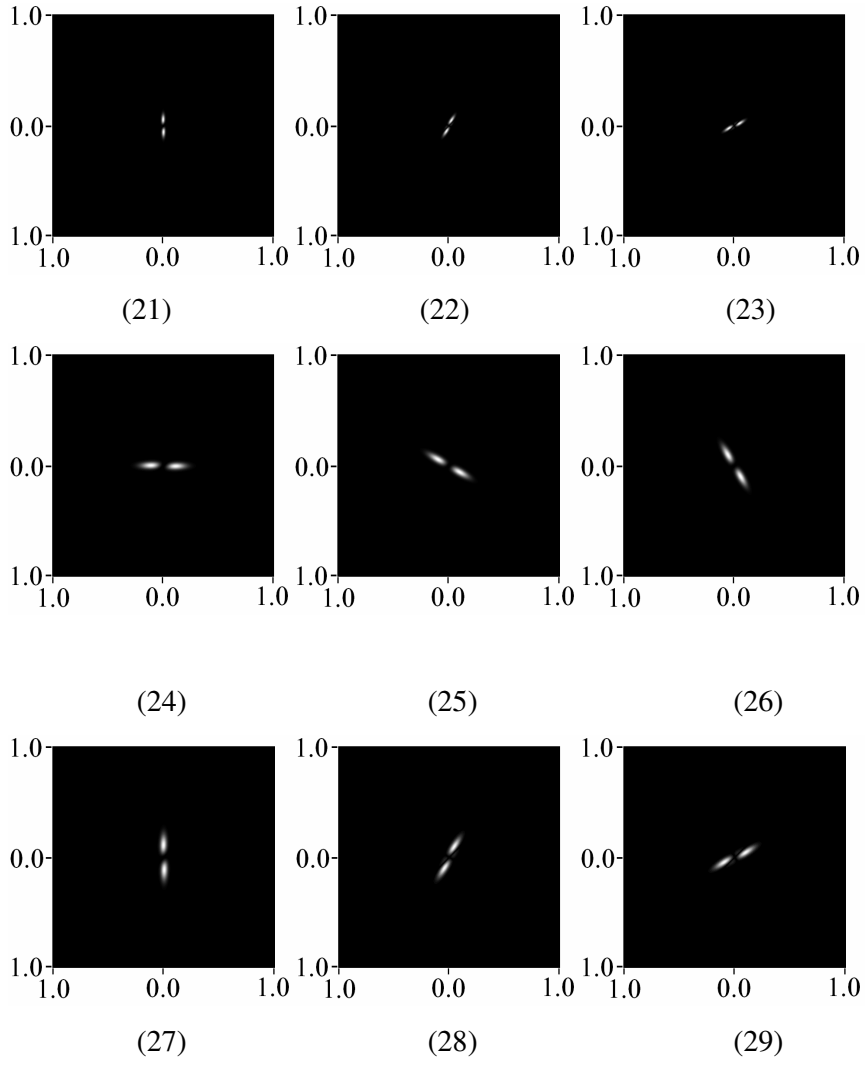


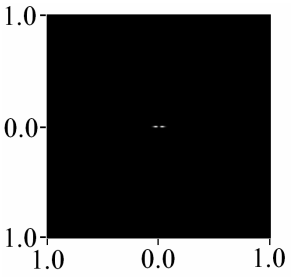
Figure 67: Edge filters in the frequency domain
(On each axis, units are shown as fraction of the Nyquist frequency)

$$G''(u,v,\theta) = \frac{(u.\cos(\theta) + v.\sin(\theta))^2 - {\sigma_r}^2}{2\pi {\sigma_r}^6} e^{-\frac{r^2}{2\sigma^2}} \qquad (7.6.7.4)$$

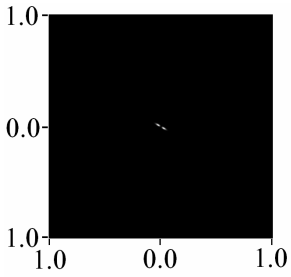
Where: $G''(u,v,\theta)$ is the Gaussian second derivative filter function,

σ_r is the standard deviation of the radius

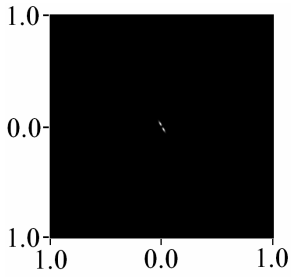
and θ is the angle, given by $\theta = \tan^{-1}(x, y)$,



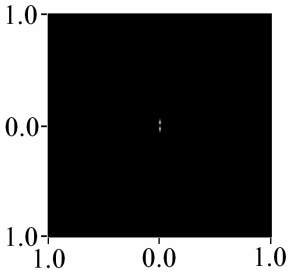
(30)



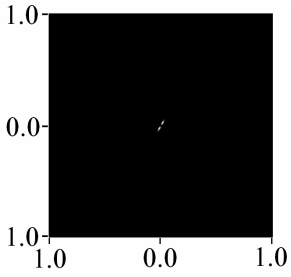
(31)



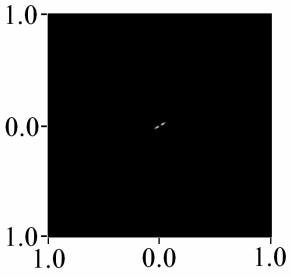
(32)



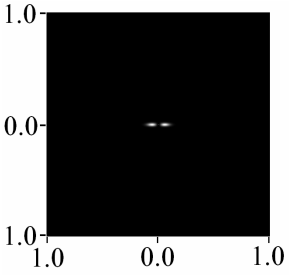
(33)



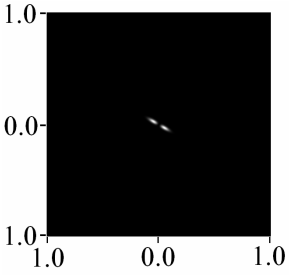
(34)



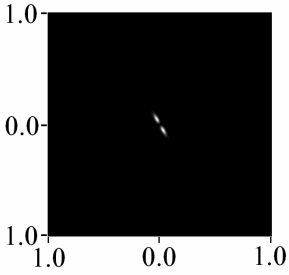
(35)



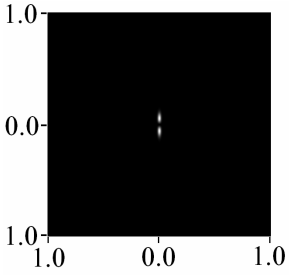
(36)



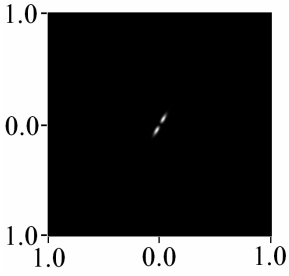
(37)



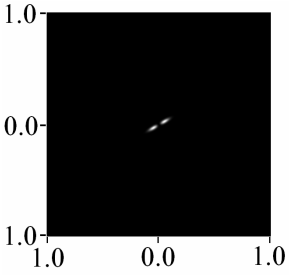
(38)



(39)



(40)



(41)

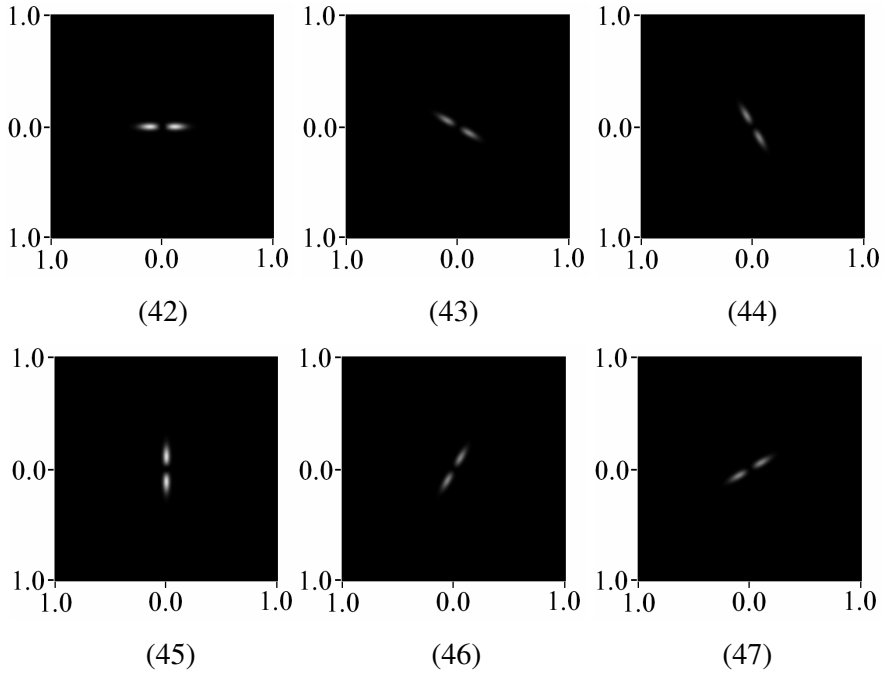


Figure 68: Bar filters in the frequency domain
(On each axis, units are shown as fraction of the Nyquist frequency)

For all of the experiments conducted by this thesis, we choose to use the standard *Leung-Malik filter bank* described by Leung-Malik [Leung2001]. Leung and Malik derived this *filter bank* from performing k-means clustering to identify the minimum number of filters required. This results in a *filter bank* comprising of forty-eight filters. For a RGB colour *filter bank*, this results in *a feature vector* size of one hundred and forty-four floating-point values.

7.6.8 The Maximum Response 4 filter bank

The *MR-4 filter bank* consists of eight filters, but unlike the previous *filter banks*, it reduces the output to four filter responses [Varma2005]. The *filter bank* consists of one *Gaussian filter*, one *Laplacian-of-Gaussian filter*, an *edge filter bank* (Gaussian first derivative) and a *bar filter bank (*Gaussian second derivative). Varma observed that the eight filter inputs could be collapsed down to four inputs while still preserving the dominant frequency response. Thus, the eight filter inputs are reduced down to four filter outputs, by keeping both the *Gaussian filter* output response and the *Laplacian-of-*

Gaussian filter output response, but only keeping the maximum response (thus the name) from the *edge filter bank* and the *bar filter bank.*

For all of the experiments conducted by this thesis, we choose to use the standard *MR-4 filter bank* described by Varma [Varma2005]. This results in a *filter bank* with four outputs for a grey scale image. For a RGB colour image, this results in *a feature vector* size of twelve floating-point values.

7.6.9 The Maximum Response 8 filter bank

The *MR-8 filter bank* consists of thirty-eight filters, but unlike the previous *filter banks*, it reduces the output to eight filter responses [Varma2005]. This *filter bank* consists of one *Gaussian filter*, one *Laplacian-of-Gaussian filter*, three banks of *edge filters* (Gaussian first derivative), with each bank having filters sensitive to each of six directions, and three banks of *bar filters* (Gaussian second derivative), also with six directions each. Varma observed that the eight filter inputs could be collapsed down to four inputs while still preserving the dominant frequency response. Thus the thirty eight filter inputs are reduced down to eight filter outputs, by keeping both the *Gaussian filter* response and *the Laplacian-of-Gaussian filter* response, but only keeping the maximum response (thus the name) from the *edge filter bank* and *bar filter bank.*

For all of the experiments conducted by this thesis, we choose to use the standard *MR-8 filter system* with eight outputs for each colour channel. This results in *a feature vector* size of twenty-four floating-point values.

7.6.10 The Polarogram

Introduced by Davis [Davis1998], the *Polarogram* is a method of analyzing the *frequency domain* of an image. In his PhD thesis, Wu used partial derivative data and we have investigated the use of this data here [Wu2003].

Because we are using a discrete FFT, the *frequency domain* image consists of a square image with an integer number of pixels in width and height. Since the *Polarogram* is a polar *histogram*, the sample area of each *Polarogram* bin consists of a wedge shaped region of the *frequency domain* image (Figure 69). Deriving the *Polarogram* from the

image of the texture in the *frequency domain* is achieved using a two dimensional lookup table the same size as the *frequency domain* image. For every pixel in the *frequency domain* image, the lookup table stores the index number of the associated *Polarogram* bin.

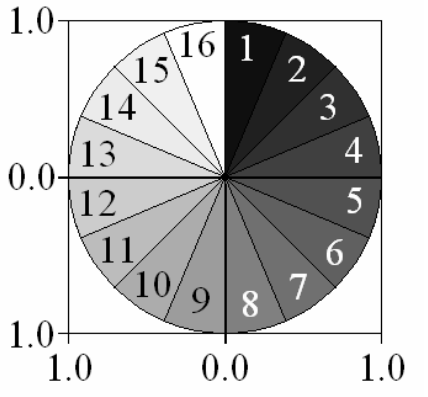


Figure 69: Example polarogram lookup table - sixteen bin polarogram

The energy level of each slice of the Polarogram is thus calculated from the sum of associated sample points. Essentially, the *Polarogram* sums together all of the energy levels for every frequency for each direction in the *frequency domain* (Figure 70).

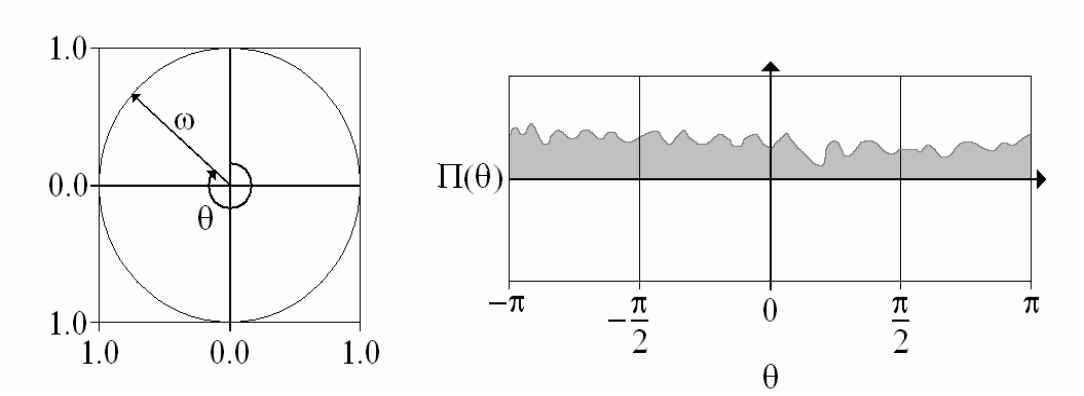


Figure 70: Calculation of the Polarogram from frequency domain data

In our implementation of the *Polarogram* filter, we construct the two dimensional look-up table as in (Figure 71). Then we transform the target image into the *frequency domain*, and accumulate each pixel in the *frequency domain* image into the *Polarogram* using this image. For this thesis, we choose to use a Polarogram consisting of 512 bins, as this guarantees that every 1 degree increment in the *frequency domain* is assigned an

individual bin. We then use the resulting *Polarogram* as a *feature vector* for *texture retrieval* purposes.

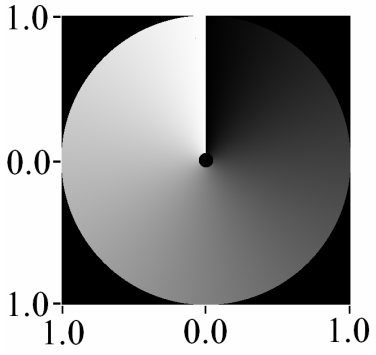


Figure 71: Polarogram mask filter for the frequency domain
(On each axis, units are shown as fraction of the Nyquist frequency)

7.6.11 The Combined filter bank

The *combined filter bank* simply combines the *feature vectors* of all of the above *filter banks* into one single large *feature vector*, with the objective being, that the increased number of filter responses should improve the *accuracy rate*. The *combined filter bank* has a data size that is the sum of all the individual *filter banks*. This results in a *feature vector* consisting of one thousand and fifty three floating-point values.

7.6.12 Summary

In section 7.6.1 to 7.6.11, we introduced a *3D surface representation* and ten methods to implement efficient *texture retrieval*. The *3D surface representation* generates a *feature vector* from each *texture image* as that is compact is size and that can be used to compare against other *feature vectors*. We have chosen to investigate the *Colour histogram* in the *spatial domain*. For the *frequency domain*, we have chosen to investigate the *ring filter bank*, the *wedge filter bank*, the *Gabor filter bank*, the *Schmid filter bank* along with the *Leung-Malik filter bank*, the *MR4 filter bank*, the *MR8 filter bank*. We have also chosen to investigate the *Polarogram*, and the *combined filter bank*.

7.7 Offsetting directionally sensitive features

While many of the *texture retrieval* methods such as the *Ring filter bank* and *Schmid filter bank* are naturally *rotation invariant,* and can thus be used to generate *rotation invariant feature vectors* directly without any further processing, there are some *texture retrieval* methods which require post-processing in order to be made *rotation invariant.* This is achieved by modifying the *feature vector* similarity operator so that instead of making a single comparison between corresponding elements of the two *feature vectors*, multiple comparisons are performed with one set of elements indexed using an offset of N. For a pair of *feature vectors* with N elements, N such comparisons will have to be performed. The lowest resulting comparison value is then returned as the result of the comparison.

7.8 Quantitative assessment of texture retrieval methods

In section 7.4, we introduced ten practical methods than can be used for *texture retrieval*. This section evaluates these methods by evaluating and comparing the *precision* and *recall* of each of these methods. The *precision* of each *texture retrieval* methods indicates the ability of that *texture retrieval* method to return only relevant database entries. The *recall* of each *texture retrieval* method indicates the ability of that *texture retrieval* method to return every relevant item to the search query.

7.8.1 Assessment results

We present the assessment results of the ten *texture retrieval* methods combined with *albedo* data (Figure 72), (Figure 73), *surface normal* ($\mathbf{n}$) data (Figure 74), (Figure 75), and the gradient $g(x, y)$ data (Figure 76) and (Figure 77).

7.8.1.1 Assessment results for albedo data

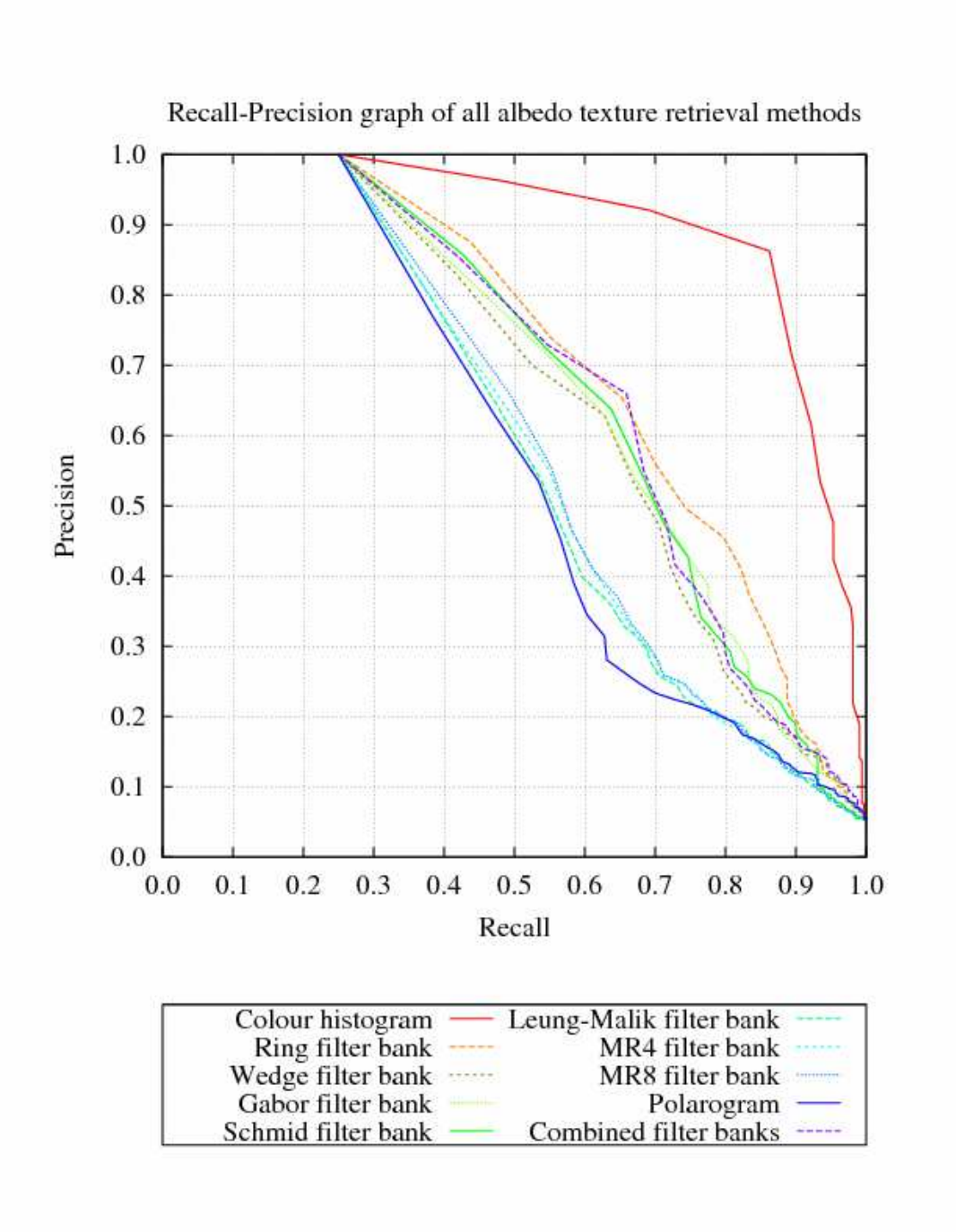


Figure 72: Recall-Precision graph of all albedo texture retrieval methods

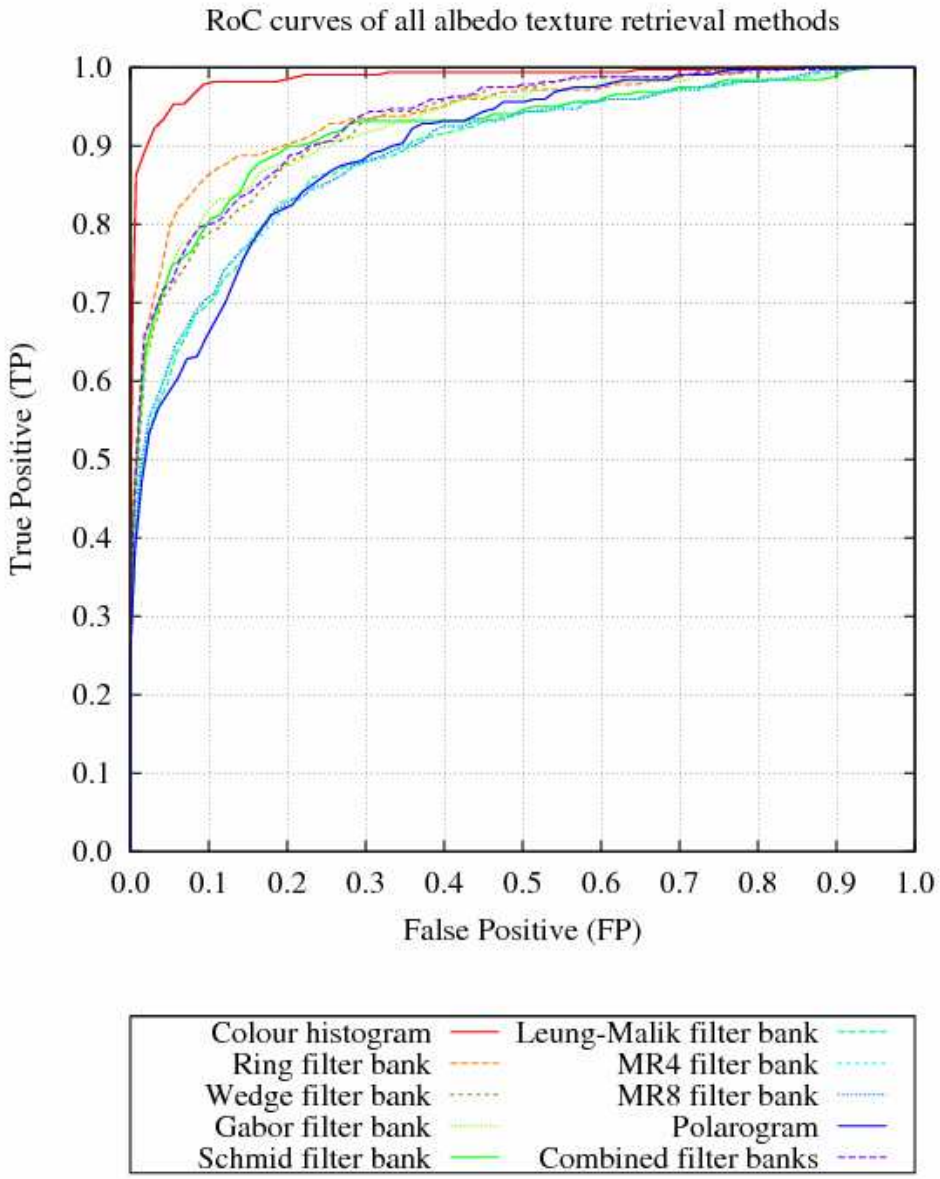


Figure 73: RoC graph of all albedo texture retrieval methods

7.8.1.2 Assessment results for surface normal data

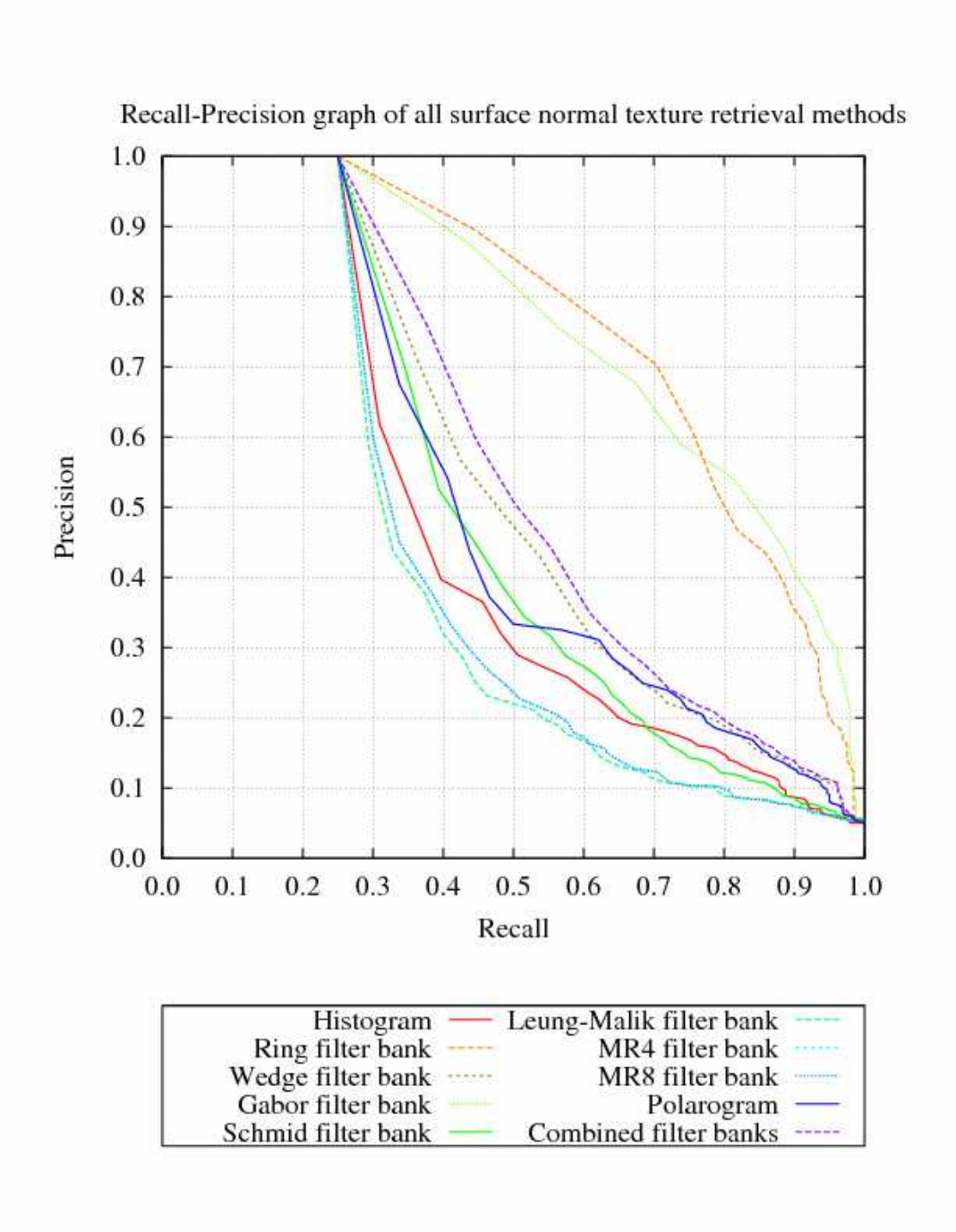


Figure 74: Recall-Precision graph of all surface normal texture retrieval methods

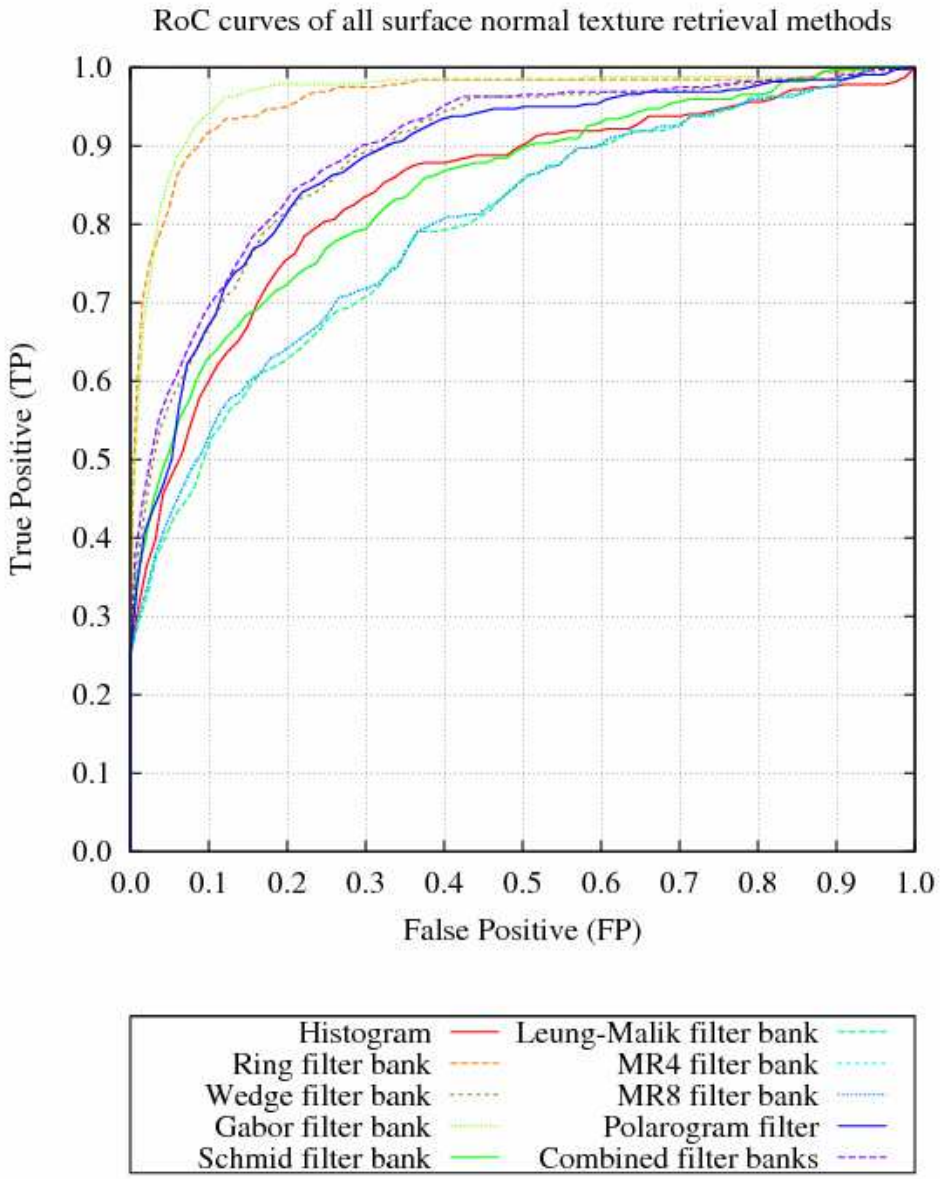


Figure 75: RoC graph of all surface normal texture retrieval methods

7.8.1.3 Assessment results for gradient data

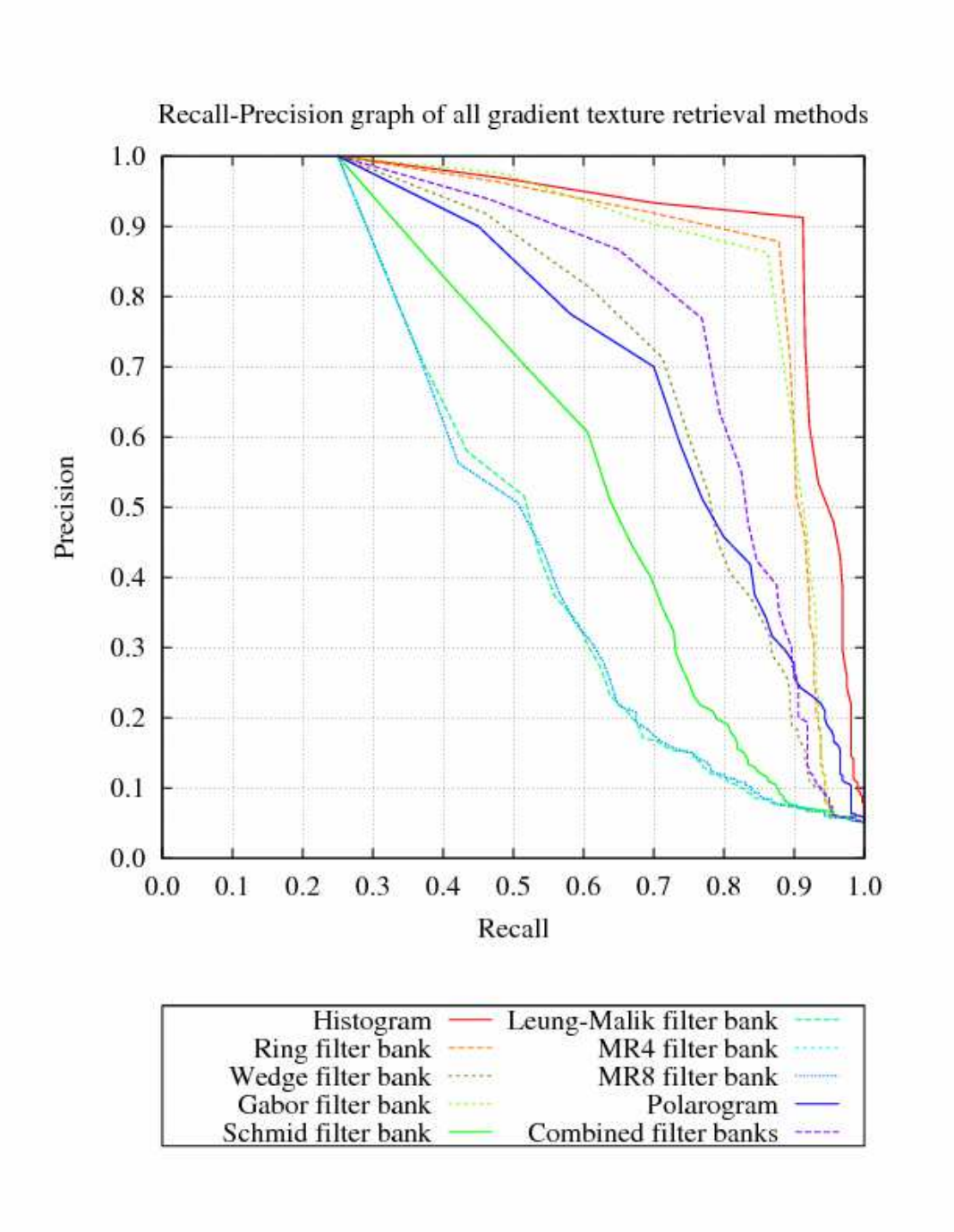


Figure 76: Recall-Precision of all gradient texture retrieval methods

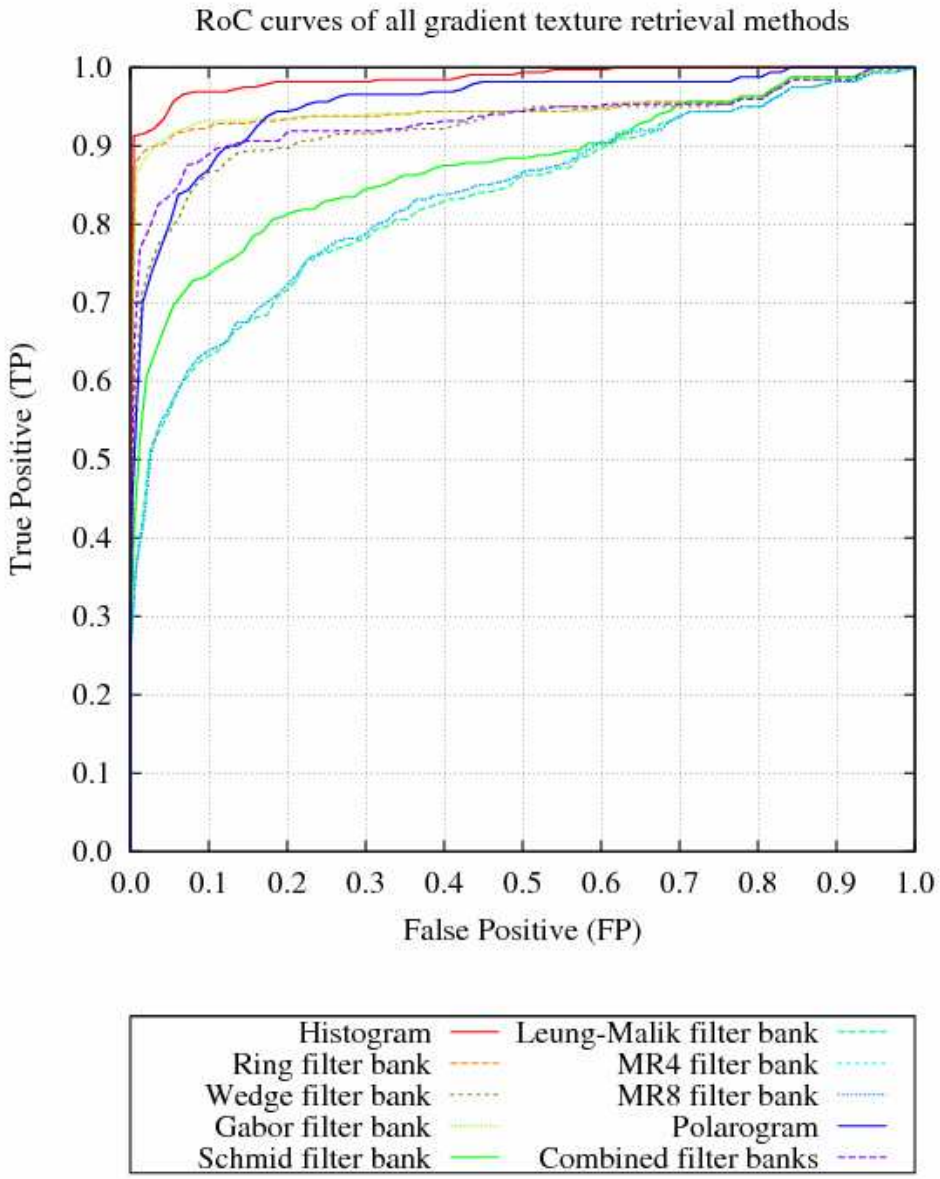


Figure 77: RoC graph of all gradient texture retrieval methods

7.8.2 Discussion of the assessment results.

This section presents, analyzes and discusses the assessment results from the different *texture retrieval* methods (the full set of results can be found in Appendix A: Texture retrieval experiment results).

7.8.2.1 Colour Data

From (Figure 72) and (Figure 73), which present the experiment results of *texture retrieval* methods combined with *albedo* data as Recall-Precision and Receiver-Operator-Characteristic (RoC) graphs, we can see that the different *texture retrieval* methods each form three distinct groups of retrieval performance. The first of these groups, the *histogram*, is clearly significantly better than all of the second order filters, suggesting that the first order statistics contain the most discriminative colour data.

The second group consists of the *wedge filter bank*, the *ring filter bank*, the *combined filter bank*, the *Schmid filter bank* and the *Gabor filter bank*. Both the *Schmid filter bank* and the *ring filter bank* are naturally *rotation invariant*. The *wedge filter bank* and the *Gabor filter bank* are both directionally sensitive but only at preselected directions. The *combined filter bank* is simply an average of all the *texture retrieval* methods used together. These all have a similar performance and all of them use a wide range of second order statistical data.

The third group consists of the *Leung-Malik filter bank*, the *MR4 filter bank*, the *MR8 filter bank* and the *Polarogram*. Because the *MR4 filter bank* and the *MR8 filter bank* are derived from the *Leung-Malik filter bank*, it is expected that these three *texture retrieval* methods have similar performance results. From the graphs we can see that they do indeed have very similar performances that are all worse than the second group. This is most likely due to the fact that they do not sample the higher frequencies that characterize the rapid colour changes present in textiles (Figure 72) and (Figure 73).

Both the *MR4 filter bank* and the *MR8 filter bank* have better performance than the *Leung-Malik filter bank*, despite having a smaller *feature vector*. The *Polarogram* had the worst performance of all the selected methods. This can be explained due to the

sampling of the entire frequency range and the use of a large number of dimensions in each *feature vector*. While other methods only have a small number of *feature vectors* (20/30 filters), the *Polarogram* has over 512 outputs. Consequently, any peaks in the *frequency domain* are spread out between many of the outputs. Examination of the resulting data reveals that for each textile sample there are less than ten peaks in the *Polarogram*, which correlate to the alignment of the weave pattern of the textile sample in the acquired images. This demonstrates that increasing directional sensitivity and increasing the size of the *feature vector* does not always improve accuracy and precision.

7.8.2.2 Micro-geometry data

(Figure 74) to (Figure 77) show the Receiver-Operator-Characteristic and Recall-Precision results for retrieval using two different types of *micro-geometry* data. The first type uses the *surface normal* information (n) directly, while the second type processes these data to provide gradient field data (g) which are theoretically free of directional artifacts. Comparing (Figure 74) and (Figure 75) with (Figure 76) and (Figure 77), we can see that the *gradient field* results are much better than those based on the *surface normal* data directly. Indeed, all of the *gradient data* recall-precision graphs are better than the corresponding *surface normal* data plots, suggesting that the removal of directional artifacts from the *micro-geometry* is very effective.

Examining the gradient results on their own (Figure 76) and (Figure 77), shows that there is no clear distinct grouping of *texture retrieval* methods, but that the curves of the *texture retrieval* methods are distributed across the graph. However, we can see that the three best performing texture retrieval methods are the *Histogram*, the *ring filter bank* and the *Gabor filter bank* and the worst three performing *texture retrieval* methods are the *Leung-Malik filter bank*, the *MR8 filter bank* and the *MR4 filter bank*. As with the colour albedo data, the poor performance of the last three feature sets can be attributed to the use of filters which do not use high frequency information. Between these two limits lie the *Polarogram*, the *Wedge filter bank* and the *combined filter bank,* with the *Wedge filter bank* having better performance than the *Polarogram*. The ranking of *texture retrieval* methods in this graph demonstrates that first order statistics are important for *micro-geometry* discrimination, but that they are not the clear cut winners

that they were with the colour *albedo* data. In particular the directionally insensitive *ring filter bank* does almost as well as the *histogram* measure here.

7.9 Conclusion

In this chapter, we selected ten methods for implementing *texture retrieval* with a textile database, which we then tested and evaluated. This forms the first component of our *3D surface representation* for the *3D visualisation* of virtual textile catalogues.

We presented a review of possible *texture retrieval* methods at the start of this chapter. Since the main objective of this chapter is to determine efficient methods of retrieval for textile catalogues, ten *rotation invariant texture retrieval* methods thave been selected. These include the *colour histogram* for *albedo* and *surface normal* data, the standard *histogram* for gradient data, and the *ring filter bank*, the *wedge filter bank*, the *Gabor filter bank*, the *Schmid filter bank*, the *Leung-Malik filter bank*, the *Maximum-Response-8 filter bank (MR-8)* and the *Maximum-Response-4 filter bank (MR-4),* the *Polarogram* and the *combined filter banks method*. We have also presented a summary of the properties of each of these *texture retrieval* methods.

We present a table listing the ranked performance of each *texture retrieval* method and texture data type in (Table 11) and a table listing the overall performance of each texture retrieval method in (Table 12). Following these two tables, we present our conclusions based upon the results of our experiments.

Rank	Albedo	Bumpmap	Gradient data
1st	Histogram	Gabor filter bank	Histogram
2nd	Ring filter bank	Ring filter bank	Ring filter bank
3rd	Combined filter bank	Combined filter bank	Gabor filter bank
4th	Gabor filter bank	Wedge filter bank	Combined filter bank
5th	Schmid filter bank	Polarogram filter bank	Wedge filter bank
6th	Wedge filter bank	Histogram	Polarogram
7th	MR4 filter bank	Schmid filter bank	Schmid filter bank
8th	Leung-Malik filter bank	MR8 filter bank	MR8 filter bank
9th	MR8 filter bank	MR4 filter bank	MR4 filter bank
10th	Polarogram	Leung-Malik filter bank	Leung-Malik filter bank

Table 11: Table of texture retrieval method rankings

Method	Ranking			Total
	Albedo	Bumpmap	Gradient field	
Gabor filter bank	4	1	3	8
Ring filter bank	2	2	2	8
Histogram	1	6	1	8
Combined filter bank	3	3	4	10
Wedge filter bank	6	4	5	15
Schmid filter bank	5	7	7	19
Polarogram	10	5	6	21
MR4 filter bank	7	9	9	25
MR8 filter bank	9	8	8	25
Leung-Malik filter bank	8	10	10	28

Table 12: Table of texture retrieval methods sorted by overall performance

We have shown for our dataset, the following:

- First order statistics in the form of histogram data provide by far the best discriminative features for colour albedo information.
- Processing the *surface normal micro-geometry* data to remove directional artifacts and produce the gradient data considerably improves performance.

- For the *micro-geometry* data, the results are less clear cut but the simple histogramming is still in the lead as a feature for use with the directionally insensitive second order information and ring filters not far behind.

We believe that our original contribution from this chapter is that this is the first time that a comparison of *rotation invariant texture classification* methods has been made with the combined use of colour (*albedo*) and 3D surface representations (*micro-geometry* represented as *normalmap* and *gradient field* data).

Chapter 8 – Conclusions and Further Work

8.1 Summary

The objectives of the research reported in this thesis were:

- Identify the most suitable method of representing the *3D surface representation* or the *micro-geometry* of textile samples acquired using economic methods.

- Identify the most suitable *rendering method* to present the acquired *3D surface representations* to the user as realistically as possible in *real-time*, using current generation *programmable graphics accelerator boards*.

- Identify the most suitable method for implementing *rotation invariant texture retrieval* based upon similarity matching of the *3D surface representation*.

We believe that the novel contributions provided by this thesis include the following:

- Creating a method of rendering *parametric surfaces* (Bézier patches) textured with the *micro-geometry* of textile samples acquired using *photometric stereo* and illuminated using both *relief mapping* and *shadow mapping* with dynamic light sources to achieve *real-time* visualisation of textile covered *3D geometric objects*.

- The investigation into the use of *rotation invariant texture retrieval* algorithms that use *normalmap* and *gradient field micro-geometry* and colour information of textile samples acquired using *photometric stereo*.

First objective

Our first objective involved identifying the most suitable method of representing the *3D surface representation* or *micro-geometry* of textile samples. To achieve this goal, we identified nine candidate methods; *General and Scattering functions*, the BSSRDF, the BTF, *Surface Light fields* and *Surface Reflectance Fields*, the BRDF/DSRF, *Polynomial Texture Maps*, *texture-mapping*, *Blinn bump-mapping*, *relief-mapping*, and *Point Set Surfaces*. We identified *relief-mapping* as the method most suitable to our needs as it matches all of our criteria for a suitable *3D surface representation*. This method consists of a pair of *texture images*, one of which defined the *albedo* image and the other defined the combined *normalmap* and *heightmap*. This imposes the requirement that a method of acquiring this *3D surface representation* is required, which is satisfied by the technique of *photometric stereo*. However, as a trade-off between economic memory usage and accuracy of *lighting model*, this method does not model the variance in reflected light due to different combinations of light source direction and camera angle. Such *lighting models* are necessary if materials such as velvet and silk are to be visualised.

Second objective

Our second objective involved identifying the most suitable *rendering method* to present the acquired *3D surface representations* to the user as realistically as possible in *real-time* using current generation *programmable graphics accelerator boards*. To achieve this objective, we conducted a survey of eight candidate *rendering methods*; shadow volumes, radiosity/discontinuity meshing, ray-tracing, scan-line algorithms, subdivision methods, *shadow mapping* and shadow fields. We identified the *shadow-mapping* method as the one most suitable to our needs as it was the only method that matched our criteria of allowing dynamic light sources to be used in *real-time* in conjunction with hardware acceleration. Thus our visualisation system consisted of the rendering of *parametric surfaces* (Bézier patches) using *relief-mapping* combined with *shadow-mapping*. This imposes the requirement that a *programmable graphics accelerator board* is used to visualize all textile samples. As it is currently implemented, our system supports dynamic light sources free to move under user

control using *relief-mapping* and *shadow-mapping* combined together. However, with more development time, it would be possible to extend this system to support physical simulation of textiles using character animation as well as using more *advanced lighting models* such as the BTF.

Third objective

Our third objective involved identifying the most suitable method for implementing *rotation invariant texture retrieval* based upon similarity matching of the *3D surface representation*. To achieve this goal, we identified ten candidate methods suitable for the *rotation invariant texture classification*. These methods included a *ring filter bank*, a *wedge filter bank*, the *Gabor filter bank*, the *Schmid filter bank*, *the Leung-Malik filter bank*, the *MR4 filter bank*, the *MR8 filter bank*, *Polarograms*, *Histograms* and all of these methods combined together. We applied each of these methods onto the *albedo*, *normalmap* and *gradient field* data generated from each textile sample and used precision-recall and receiver-operator-characteristic graphs to analyze the results. After analyzing the results, we identified the *colour histogram*, the *Gabor filter bank* and the *ring filter bank* as the most suitable methods for the implementation of a virtual textile database. The use of the *Gabor filter bank* and the *ring filter bank* has the consequence of requiring that the discrete FFT is used to generate *feature vectors* suitable for use with *texture retrieval*. While we were able to implement the core functionality of a *texture retrieval* database, the implementation of an interactive user interface goes well beyond the scope of of this PhD thesis, and was not addressed. Research into this area is ongoing.

Contribution

In terms of achieving the original objectives of this thesis, we believe we have achieved the following.

We have achieved the objective of identifying a suitable *3D surface representation* that can be used to represent the *colour* and *micro-geometry* information of textile samples suitable for use with *3D visualisation* and *texture retrieval*, given the constraints of economic memory usage versus complexity of *lighting model*. We have also achieved the objective of combining together the *real-time rendering* of the *macro-structure* in the form of Bézier surfaces and the *micro-structure* in the form of *surface normal* and *displacement maps 3D geometric models* using current *programmable graphics accelerator boards* in order to implement *3D visualisation* of textile samples using a variety of 3D forms. This system utilizes *shadow-mapping* for the *macro-structure* and *relief-mapping* for the *micro-structure.*

We have also achieved the objective of identifying the most suitable *texture retrieval* methods to generate *rotation invariant feature vectors* using the data represented by the *3D surface representation,* which consists of the *albedo*, *normalmap* and *gradient field*. Our feature vectors are novel in that they are generated from the combined use of both *albedo* and *gradient field* data, to the best of our knowledge has not been combined together before for the purpose of *texture retrieval*.

We concluded that for our test set, first order statistics in the form of *histogram* data clearly provide the best performance when using colour *albedo* data. When using *micro-geometry* data, computational processing to remove the directional artifacts present in the partial derivatives significantly improves performance, and while the first order statistics are again important and provide good retrieval performance, two of the second order features (the *ring filter bank* and the *Gabor filter bank*) also gave good performances.

8.2 Further Work

In the previous section, we described the original objectives of this thesis, how far each objective was achieved and what issues remained unresolved. In this section we describe in greater detail these outstanding issues and how they can be resolved. We can classify these outstanding issues into three categories:

- Visualisation
- Front-end user interface
- Back-end texture retrieval system

For the visualisation component of our system, there are several outstanding issues and improvements that could be made. These include a more advanced *3D surface representation* to take into the change of appearance of the textile sample due to changing lighting and camera angles, such as the BTF (Bidirectional Texture Function). This would require a modification of the *photometric stereo* acquisition process to capture multiple images of the textile samples from varying camera angles and light source directions. This would also require using one of the many methods of compressing the hundreds of megabytes of raw data into one or more *texture images* using statistical analysis methods such as *spherical harmonics*, single value decomposition or principal component analysis [Filip2008b]. The resulting *texture images* would then be used to implement the BTF.

Another improvement to the visualisation system would be the use of real-world *3D geometric data* generated from professional CAD systems such as CATIA for the manufacturing industry. This would allow users to visualize textiles as they would be seen on real-world furniture and accessories. The use of more realistic *3D geometric models* could also be expanded to include animated human characters for the visualisation of clothing. Using a physics system running under an environment such as CUDA, PhysX, or APEX would allow for the simulation of the draping, folding and wrinkling of textures in *real-time*.

For the *texture retrieval* system, there are also several improvements that could be made to both the user interface and retrieval system. To make the user interface of the virtual

textile database catalogue more intuitive, simple visual effects such as turning pages and ring binder effects could be used. For dedicated systems in a public space the use of touch screen technology using tablet PC's or kiosk type displays could also be utilized. Such a system would allow the user to issue combinations of "find similar textures" and "find similar colours" requests.

Another improvement to the operation of the user interface would be to support relevance feedback. Instead of simply sending each query to the database engine and returning the set of results directly back to the user, relevance feedback performs each query in two stages. In the first stage, the query is sent to the database engine as before. However when the results are analyzed, they are compared to identify common terms to be added to the original search query. The new modified search query is then resubmitted to the database engine and the results returned to the user. To implement this for use with textile database systems, this would require an analysis of the *feature vectors* associated with each returned result. This could be achieved by calculating the mean and standard deviation of each *feature vector* field and then modifying the search engine comparison algorithm accordingly.

With the actual texture retrieval search engine there are also several improvements that could be made. The first would be to integrate the research conducted by this thesis into an existing framework for content rich data such as Ferret in order to provide a complete information retrieval system with a web based user-interface. Another improvement that we believe we could make, would be to make use of multi-processor and multi-core technology to speed up database search queries. This could be achieved through suitable API's such as MPI or OpenMP, with communication between the front end user interface and back end database engine handled using an internet web page browser.