Contents

Vertex Processor

Role

A vertex shader is a graphics processing function used to add special effects to objects in a 3D environment by performing mathematical operations on the objects' vertex data. Each vertex is defined by many different variables.

For instance, a vertex is always defined by its location in a 3D environment using the x-, y-, and z-coordinates. Vertices may also be defined by colors, textures, and lighting characteristics.

Vertex Shaders do not actually change the type of data; they simply change the values of the data, so that a vertex emerges with a different color, different textures, or a different position in space.

Vertex shaders are run once for each vertex given to the graphics processor. The purpose is to transform each vertex's 3D position in virtual space to the 2D coordinates at which it appears on the screen (as well as a depth value for the Z-buffer).

Block Diagram

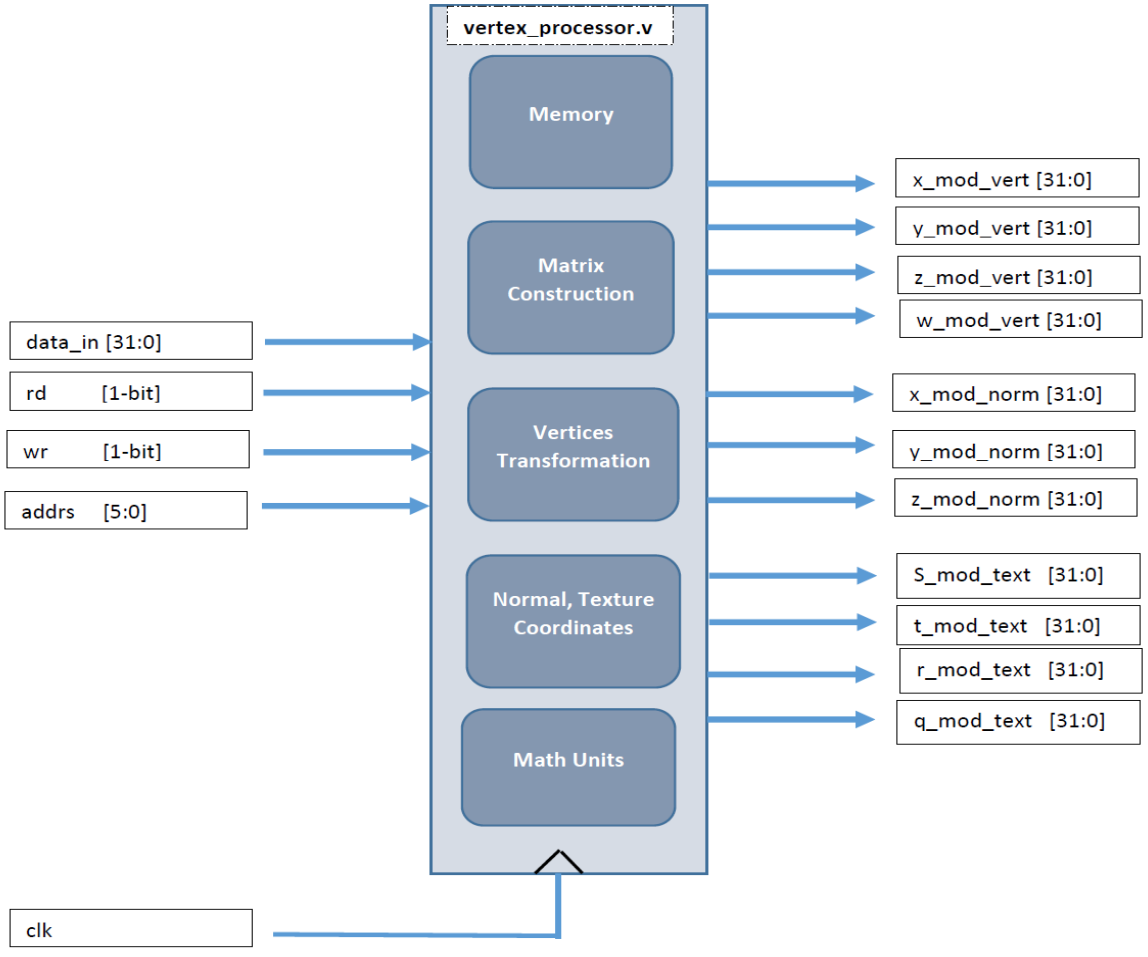


Figure 1: Block diagram of vertex processor

IO's

Signal Name	Description	Size[bits]
Inputs		
data_in	the input data from GPU Memory "Main Memory	32
wr	Write Signal for Writing into Memory	1
rd	Read Signal for Reading from Memory	1
addrs	Address Signal for accessing words in memory	6
Outputs		
x_mod_vert	The modified Positional Parameter X-coordinates of vertex	32
y_mod_vert	The modified Positional Parameter Y-coordinates of vertex	32
z_mod_vert	The modified Positional Parameter Z-coordinates of vertex	32
w_mod_vert	The modified Positional Parameter W-coordinates of vertex	32
x_mod_norm	The Modified Normal X-Coordinates	32
y_mod_norm	The Modified Normal X-Coordinates	32
z_mod_norm	The Modified Normal Z-Coordinates	32
s_mod_text	The Modified Texture S-Coordinates	32
t_mod_text	The Modified Texture T-Coordinates	32
r_mod_text	The Modified Texture R-Coordinates	32
q_mod_text	The Modified Texture Q-Coordinates	32

Internal Blocks

Memory, Matrix Construction, Vertices Transformation, "Normal, Texture Coordinates", Math Units.

Important Notes from OpenGL and Design constraints

- We support 3D.
- Input Vertices are in Object space.
- Input Vertices Data: Positional Parameter + Color +Texture +Normal.
- If texture Units supported, we need at least 2-Texture Units as Specified in OpenGL 1.1.12.
- All Blocks including top-level blocks and internal block are sequential blocks.
- All computations are performed in fixed-point format and no overflow.
- Division by zero mustn't lead to GL-interpretation or termination, but lead to un specified result.
- If two signals, including internals, inputs and outputs ports, have the same name, this means that they are connected.
- **"vert"** means vertex. A vertex defines a point, an endpoint of an edge, or a corner of a triangle where two edges meet as specified in OpenGL.
- The size of some signals are suggested such as row1_mv is the first row of model view matrix and may be updated according to creation of math units Block

Description of Internal Blocks

Memory

Role

Store and Pass required data for vertex processor from main memory.

IO'S

Signal Name	Description	Size[bits]
Inputs		
data_in	the input data from GPU Memory "Main Memory	32
wr	Write Signal for Writing into Memory	1
rd	Read Signal for Reading from Memory	1
addrs	Address Signal for accessing words in memory	6
Outputs		
vert_x	Required data relates to X-Coordinates such Positional parameter, sx "Scale value of X",rx ,tx and any other data except any cleared output ports from memory such as **vert_norm_x** which specify the value of norm for x_coordinate, **so vert_x doesn't pass this value.**	32
vert_y	The same definition and constraints of vert_x, but this relates to Y-Coordinates.	32
vert_z	The same definition and constraints of vert_x , but this relates to Z-Coordinates	32
w	Required fourth dimension for translation.	32
vert_norm_x	Normal of vertex in X-Coordinates	32
vert_norm_y	Normal of vertex in Y-Coordinates	32
vert_norm_z	Normal of vertex in Z-Coordinates	32
vert_text{0 **or** 1}_s	Texture coordinate s of vertex."0" for first texture unit."1" for second texture unit.	32
vert_text{0 **or** 1}_t	Texture coordinate t of vertex	32
vert_text{0 **or** 1}_r	Texture coordinate r of vertex	32
vert_text{0 **or** 1}_q	Texture coordinate q of vertex	32
vert_color	Color of vertex	32
ctrl	control bits	32
sin_theta	Sine of angel θ for rotation block	32
cos_theta	Cosine of angel θ for rotation block	32
clk	Clock Signal	1

Block Diagram of Memory

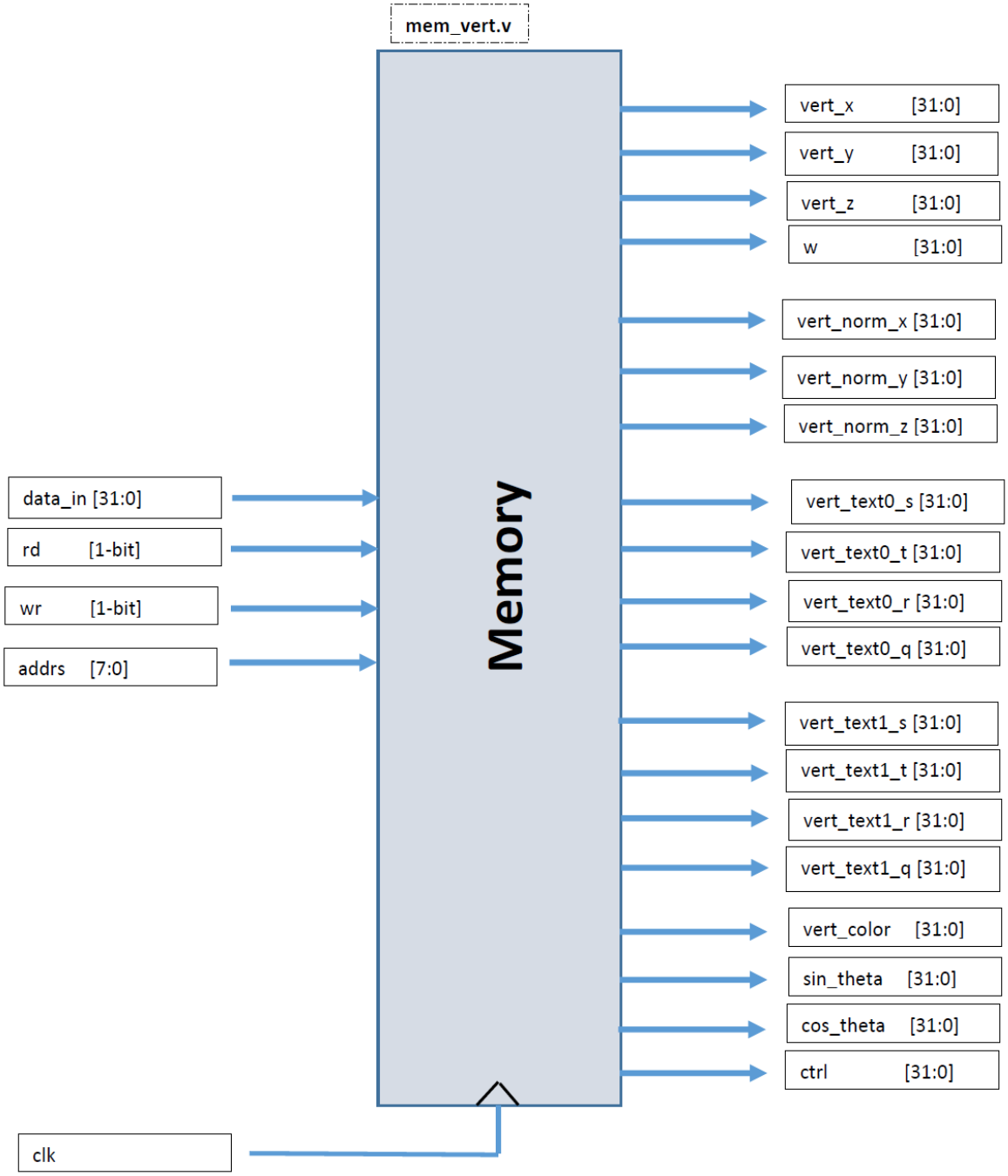


Figure 2: Vertex processor memory block diagram

Matrix Construction

Role

Prepare Matrices such as Model, View, Texture and Projection Matrices.

Scenario: First Prepare Model Matrix by multiplication between 3-matrices,but note the sequence for construction of this matrix by Multiply the Translate matrix [4x4] x Rotate matrix [4x4] and then multiply the result by scale matrix [4x4]. Translate →Rotate→Scale.

Second: Multiply the model matrix by the view matrix to construct the model-view matrix.

Note: Values of scaling and rotation and translation (Sx,Sy,Sz),(Rx,Ry,Rz),(Tx,Ty,Tz) will be passed through vert_x,vert_y,vert_z which are the output of vertex memory.
(tx,ty,tz) are just values to indicate the coordinates needed for translation and will be recognized from control register that pass through control unit in matrix construction block, **but not** the actual values of translation as these values will be passed through vert_x, vert_y, vert_z. The same Concept for Rotation and Scaling Blocks. en_{t,r,s} : For enabling the blocks of translation ,rotation ,scaling and also recognized from control register.

IO'S

Signal Name	Description	Size[bits]
Inputs		
vert_x	Required data relates to X-Coordinates such Positional parameter, sx "Scale value of X",rx ,tx and any other data stored in memory of vertex required such as Eye direction and position and so on. **Note**: The same constraints existed in memory block.	32
vert_y	The same definition and constraints of vert_x, but this relates to Y-Coordinates.	32
vert_z	The same definition and constraints of vert_x, but this relates to Z-Coordinates.	32
ctrl	Control Bits	32
Outputs		
row1_mv	The first row of model-view matrix	128
row2_mv	The second row of model-view matrix	128
row3_mv	The third row of model-view matrix	128
row4_mv	The fourth row of model-view matrix	128
row1_p	The first row of projection matrix	128
row2_p	The second row of projection matrix	128
row3_p	The third row of projection matrix	128
row4_p	The fourth row of projection matrix	128
row1_text0	The first row of texture matrix for 1^{st} texture unit	128
row2_text0	The second row of texture matrix 1^{st} texture unit	128
row3_text0	The third row of texture matrix 1^{st} texture unit	128
row4_text0	The fourth row of texture matrix 1^{st} texture unit	128
row1_text1	The first row of texture matrix for 2^{nd} texture unit	128
row2_text1	The second row of texture matrix 2^{nd} texture unit	128
row3_text1	The third row of texture matrix 2^{nd} texture unit	128
row4_text1	The fourth row of texture matrix 2^{nd} texture unit	128
clk	clock signal for all **internal blocks**	1

Internal Blocks

1- Translate Block: To construct Translation Matrix.
2- Rotate Block: To construct Rotation Matrix.(sin_theta,cos_theta) will be passed from main memory through their output ports.(cw_ccw) to indicate the direction of rotation.
3- Scale Block: Construct Scaling matrix.
4- View Block: Construct view matrix.
5- Projection Block: Construction of one type of projection matrix according to these signals enabling p_ortho, p_persp.
6- Texture Unit: Construct texture matrix, en_text known from control register to enable texture unit.
7- Control Unit: The input is the control register in vertex memory and produce output signals for enabling other block inside matrix construction block.
8- Matrix Multiplier: is an instantiation from math units block for matrix multiplier unit

Matrix Construction block

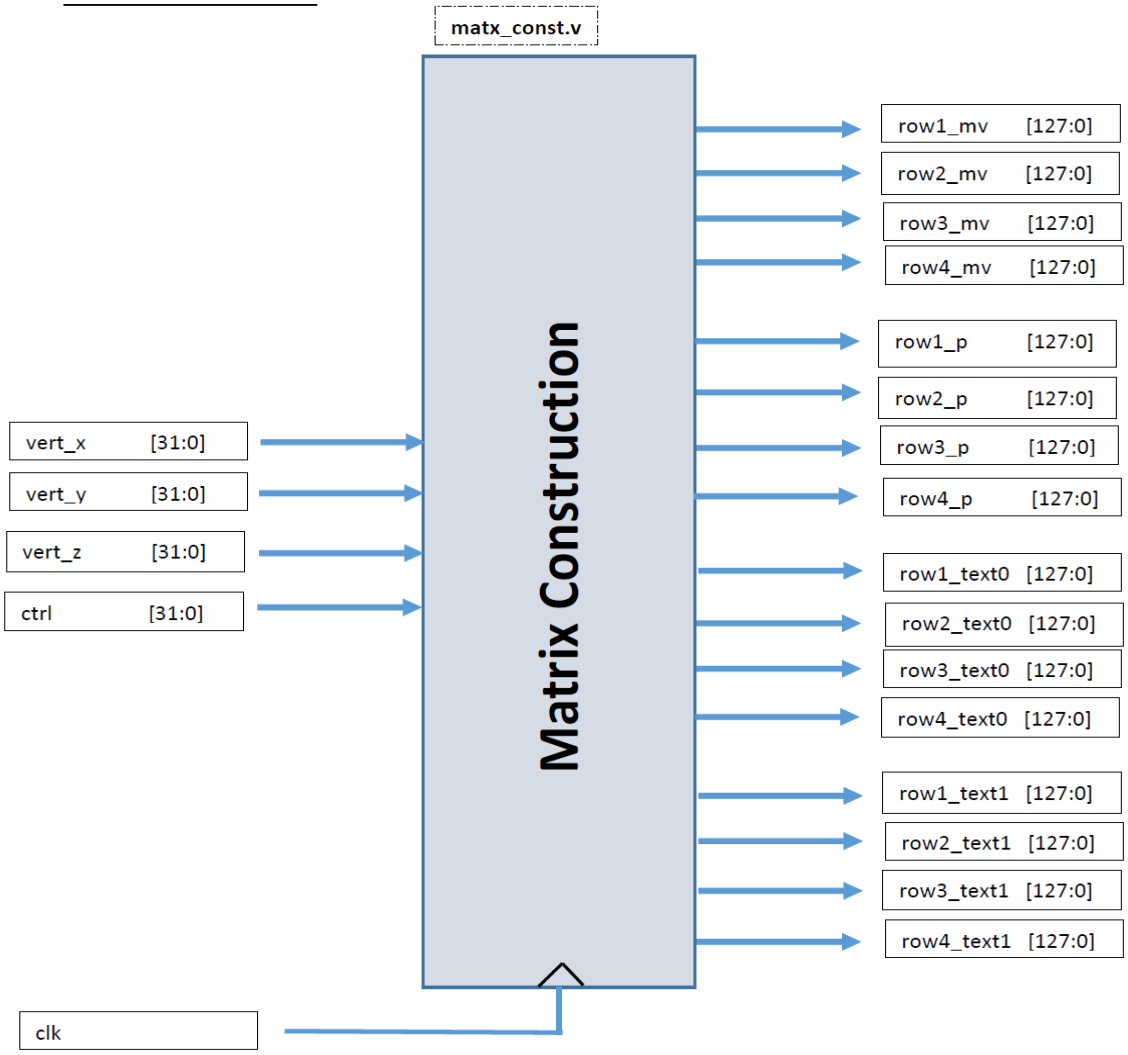
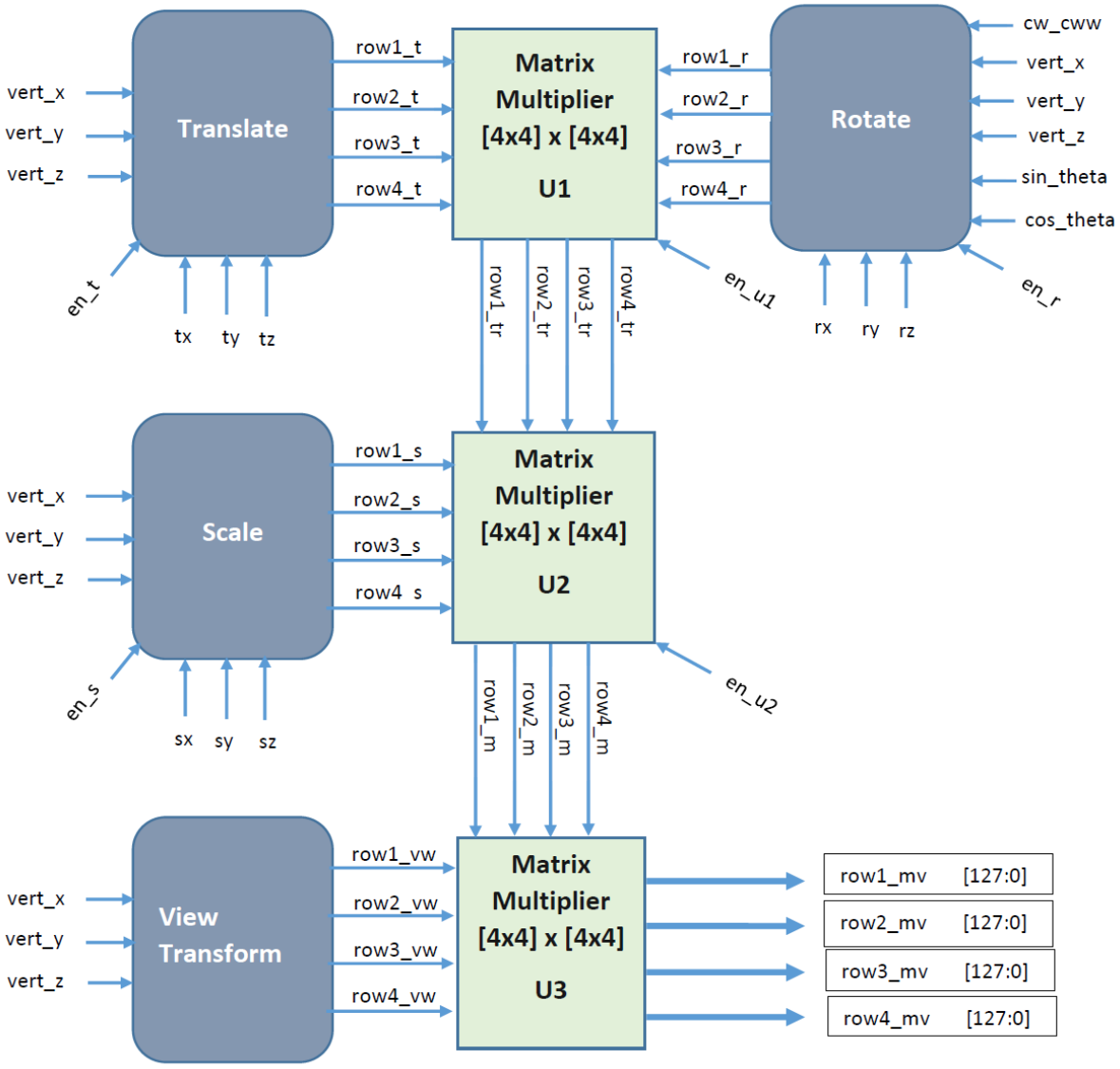
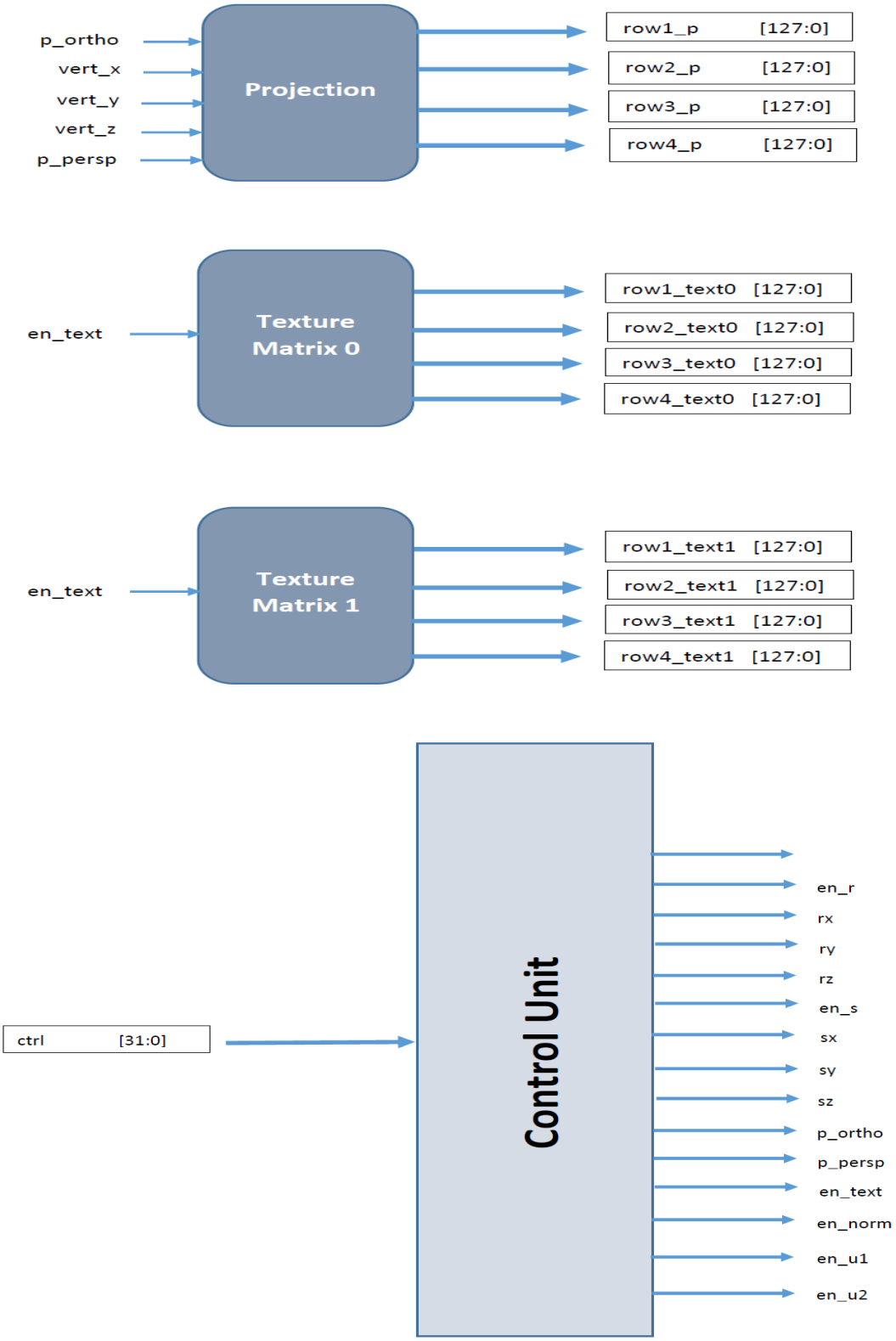


Figure 3: Matrix constructor

Interior view of Matrix Construction Block




p_ortho
vert_x
vert_y
vert_z
p_persp
Projection
row1_p [127:0]
row2_p [127:0]
row3_p [127:0]
row4_p [127:0]
en_text
Texture Matrix 0
row1_text0 [127:0]
row2_text0 [127:0]
row3_text0 [127:0]
row4_text0 [127:0]
en_text
Texture Matrix 1
row1_text1 [127:0]
row2_text1 [127:0]
row3_text1 [127:0]
row4_text1 [127:0]
ctrl [31:0]
Control Unit
en_r
rx
ry
rz
en_s
sx
sy
sz
p_ortho
p_persp
en_text
en_norm
en_u1
en_u2

Math units

Role

Prepare required Math units such as matrix multiplier.

Internal Blocks

1- **Cosine-Sine Block:** calculate the sine and cosine of input theta.
2- **Matrix Multiplier:** support different matrix multiplier units.

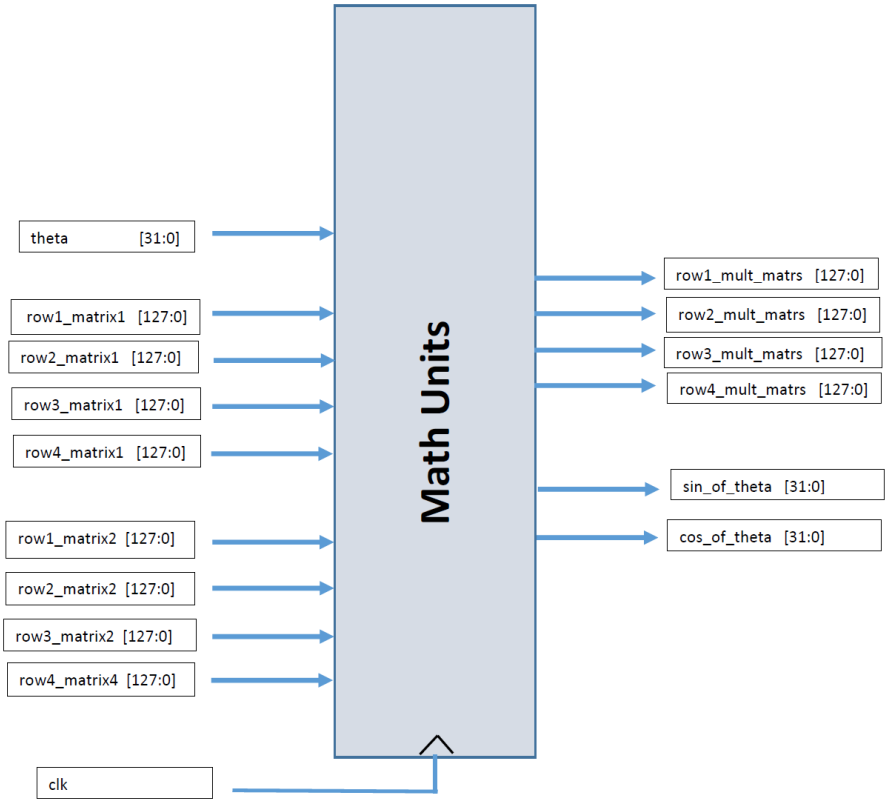
IO'S

Signal Name	Description	Size[bits]
Inputs		
theta	Angel of Rotation	32
row1_matrix1	Row1 of first matrix	128
row2_matrix1	Row2 of first matrix	128
row3_matrix1	Row3 of first matrix	128
row4_matrix1	Row4 of first matrix	128
row1_matrix2	Row1 of Second matrix	128
row2_matrix2	Row2 of Second matrix	128
row3_matrix2	Row3 of Second matrix	128
row4_matrix2	Row4 of Second matrix	128
Outputs		
sin_of_theta	Calculated Sine of theta	32
cos_of_theta	Calculated Cosine of theta	32
row1_mult_matrs	The result row1 of matrix multiplier unit	128
row2_mult_matrs	The result row2 of matrix multiplier unit	128
row3_mult_matrs	The result row3 of matrix multiplier unit	128
row4_mult_matrs	The result row4 of matrix multiplier unit	128

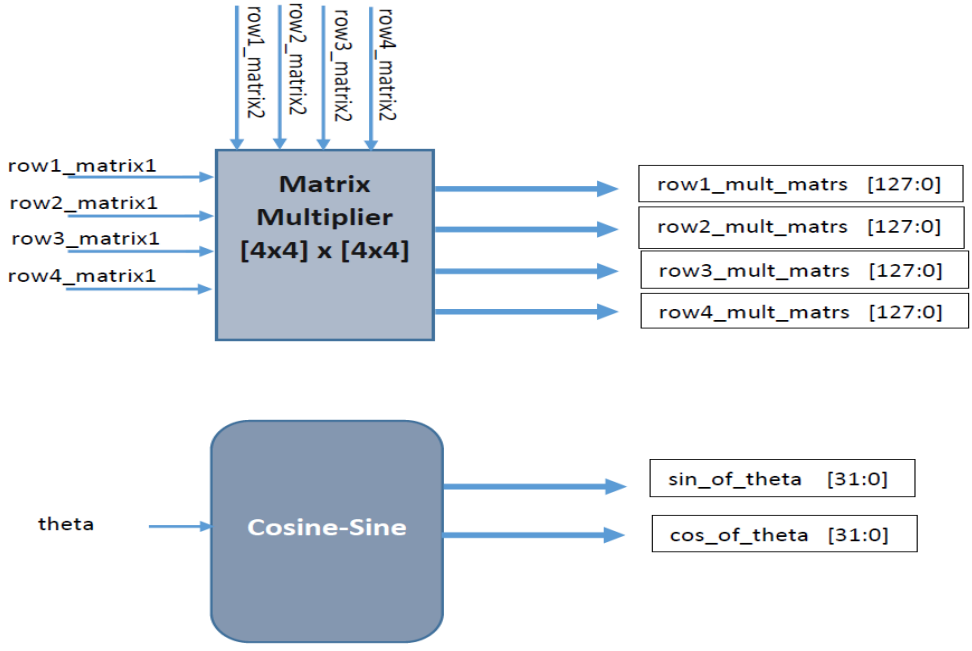
Notes

This approach is suggested for our design . Also Suggested Matrix unit can support different types. Sin_of _theta is **different** in only naming from that signal sin_theta as the data calculated from Cosine-Sine Block will first passed to data_in of vertex processor for storing in our memory and then pass this signal to sin_theta port. sin_of_theta →data_in "vertex memory"→ sin_theta.

Block Diagram



Interior View of Math units



Vertices Transformation

Role

Transform each vertex's 3D"Object Coordinate" to Clip Coordinate, and then perform Clipping Algorithm.
Scenario: First Get required Matrices from "Transformation" Matrices Construction Block".
Second Multiplication between (vertex in object coordinates and model-view matrix), then multiply the result by projection matrix. Then apply the result to the clipping stage.

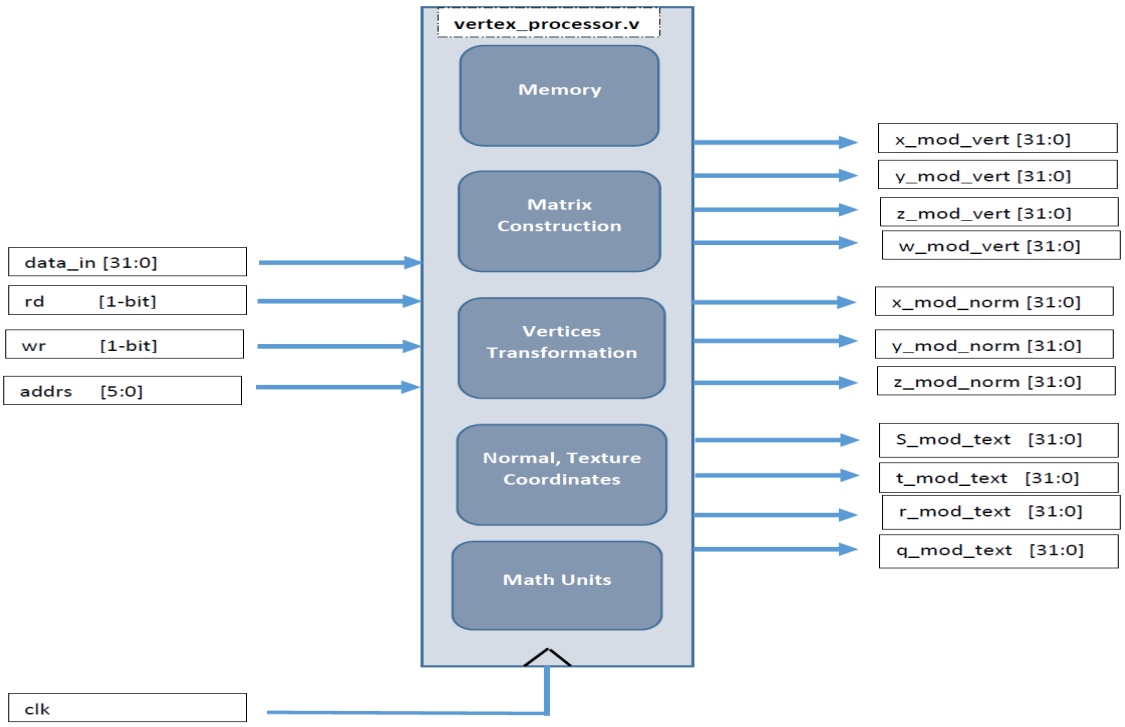
IO'S

Signal Name	Description	Size[bits]
Inputs		
vert_x	Here is the positional parameter of vertex in X-coordinates **"Object Space"**	32
vert_y	Here is the positional parameter of vertex in Y-coordinates **"Object Space"**	32
vert_z	Here is the positional parameter of vertex in Z-coordinates **"Object Space"**	32
row1_mv	The first row of model-view matrix	128
row2_mv	The second row of model-view matrix	128
row3_mv	The third row of model-view matrix	128
row4_mv	The fourth row of model-view matrix	128
row1_p	The first row of projection matrix	128
row2_p	The second row of projection matrix	128
row3_p	The third row of projection matrix	128
row4_p	The fourth row of model-projection matrix	128
Outputs		
x_mod_vert	The modified Positional Parameter X-coordinates of vertex	32
y_mod_vert	The modified Positional Parameter Y-coordinates of vertex	32
z_mod_vert	The modified Positional Parameter Z-coordinates of vertex	32
w_mod_vert	The modified Positional Parameter W-coordinates of vertex	32

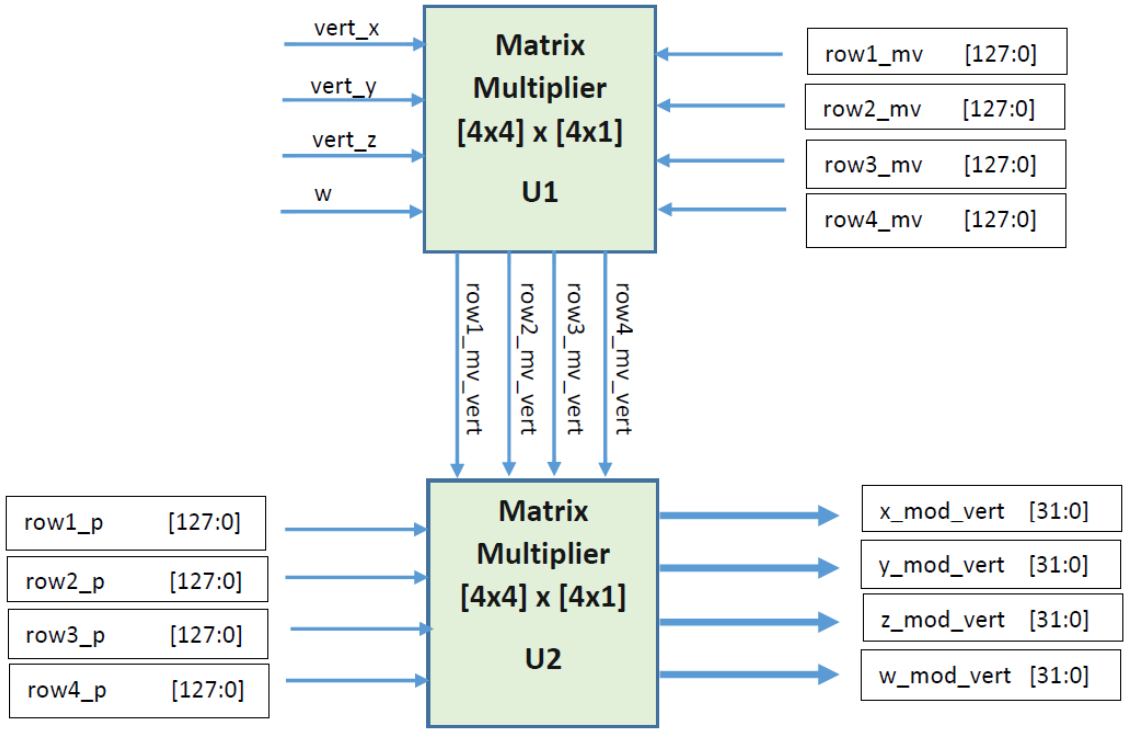
Internal Blocks

- Only two multiplier Units, but multiply [4x4]x[4x1]

Vertices Transformation Block



Interior view of vertices transformation block



Normal, Texture Coordinates

Role

Get the modified normal coordinates for lighting stage and get the modified texture coordinates

Note: The steps required for normal, Texture block will be supported in a pdf file from architecture team. **Also**, the unit used for Texture unit0 will be instantiated, but different data enters which required for texture unit1.

IO'S

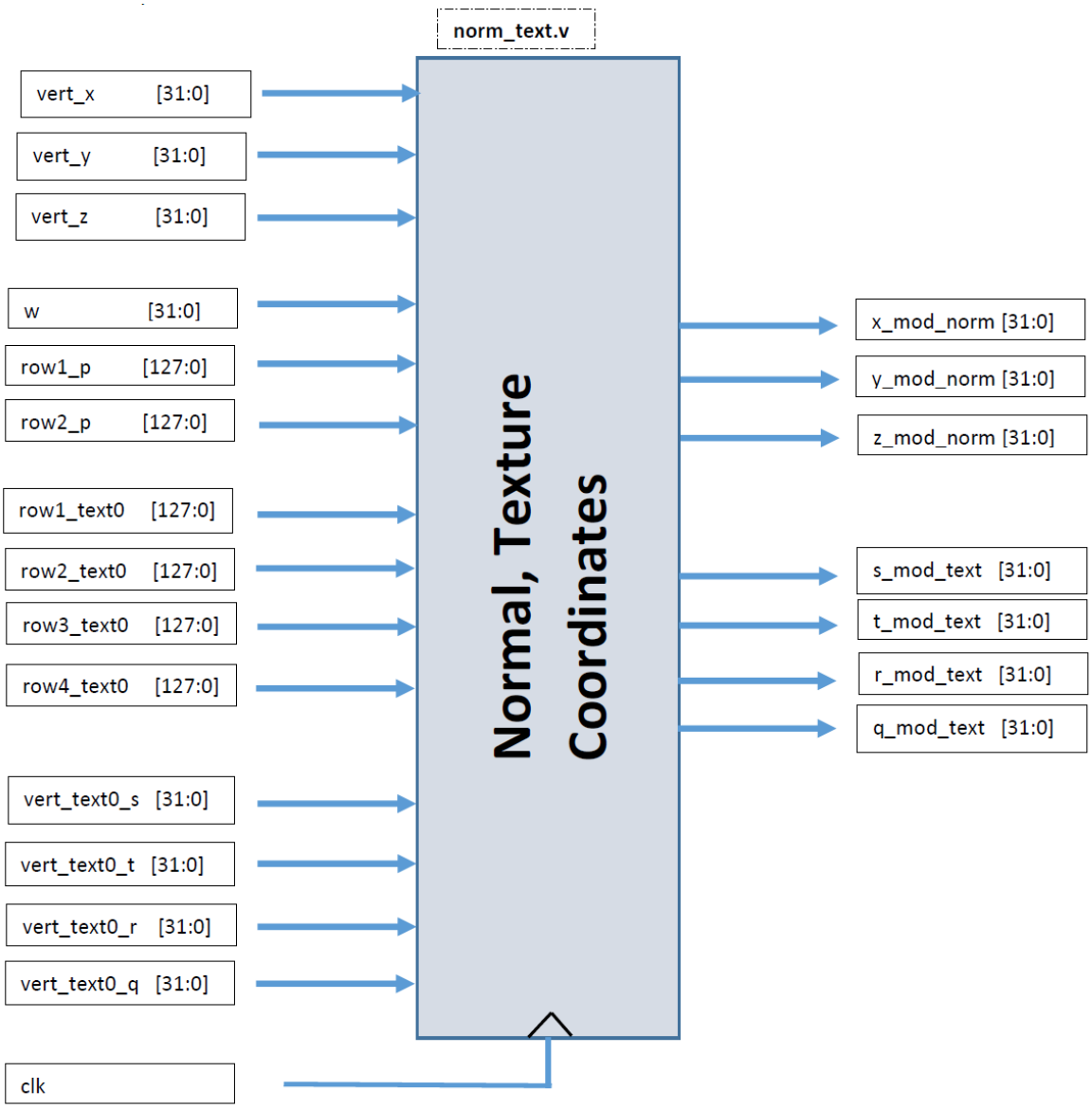
Signal Name	Description	Size[bits]
Inputs		
vert_norm_x	Normal of vertex in X-Coordinates	32
vert_norm_y	Normal of vertex in Y-Coordinates	32
vert_norm_z	Normal of vertex in Z-Coordinates	32
row1_mv	The first row of model-view matrix	128
row2_mv	The second row of model-view matrix	128
row3_mv	The third row of model-view matrix	128
row1_text0	The first row of texture matrix for 1^{st} texture unit	128
row2_text0	The second row of texture matrix 1^{st} texture unit	128
row3_text0	The third row of texture matrix 1^{st} texture unit	128
row4_text0	The fourth row of texture matrix 1^{st} texture unit	128
vert_text0_s	Texture coordinate s of vertex."0" for first texture unit."1" for second texture unit.	32
vert_text0_t	Texture coordinate t of vertex	32
vert_text0_r	Texture coordinate r of vertex	32
vert_text0_q	Texture coordinate q of vertex	32
Outputs		
x_mod_norm	The Modified Normal X-Coordinates	32
y_mod_norm	The Modified Normal X-Coordinates	32
z_mod_norm	The Modified Normal Z-Coordinates	32
s_mod_text	The Modified Texture S-Coordinates	32
t_mod_text	The Modified Texture T-Coordinates	32
r_mod_text	The Modified Texture R-Coordinates	32
q_mod_text	The Modified Texture Q-Coordinates	32

Internal Blocks

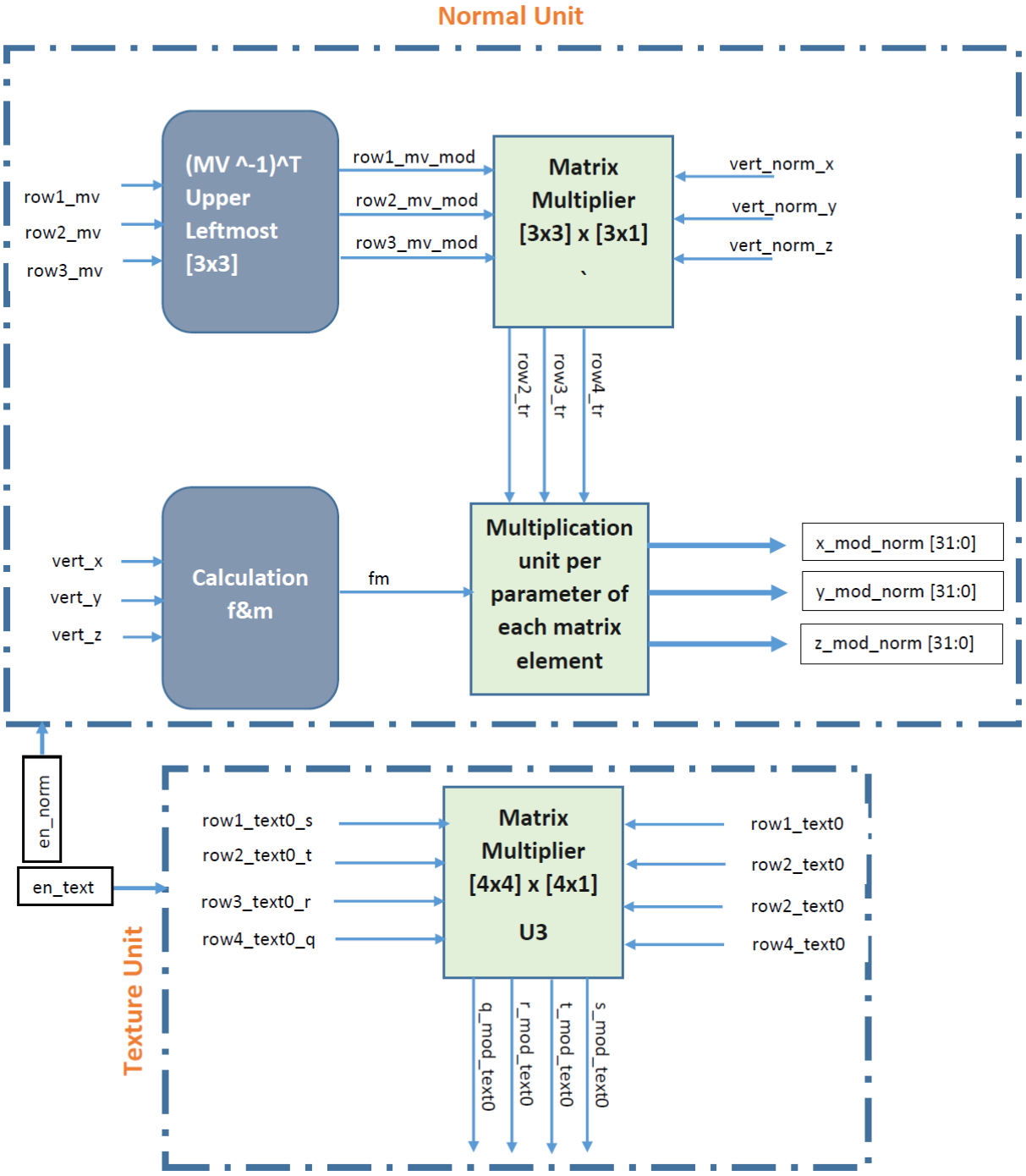
These blocks will be declared in another documentation and will be supported .In google drive references →file: Vertex Processor(moaz).pdf

1. **(MV^-1)^T:** Block used to get the upper leftmost model-view matrix.
2. **Calculation f&m:** Calculating f&m parameters used to get the modified normal.
3. **Matrix Multiplier:** Matrix Multiplier [3x3]x[3x1].
4. **Multiplication Unit:** Multiply each matrix element by the result of calculation of f&m.

Normal, Texture Coordinates Block



Interior View of Normal, Texture Coordinates



Clipping Stage

Role

Primitives are clipped to the clip volume .Primitives are defined by a group of one or more vertices. Then multiply the result by the viewport matrix to get vertices in screen space.

Block Diagram

Signal Name	Description	Size[bits]
Inputs		
x_mod_vert	The modified Positional Parameter X-coordinates of vertex	32
y_mod_vert	The modified Positional Parameter Y-coordinates of vertex	32
z_mod_vert	The modified Positional Parameter Z-coordinates of vertex	32
w_mod_vert	The modified Positional Parameter W-coordinates of vertex	32
l	World_Space_Left_Side(L)	32
r	World_Space_Right_Side(R)	32
t	World_Space_Top_Side(T)	32
b	World_Space_Bottom_Side(B)	32
n	World_Space_Near_Side(N)	32
f	World_Space_Far_Side(F)	32
nx	Screen_Width(Nx)	32
ny	Screen_Height(Ny)	32
Outputs		
x_image	Vertex position X-Coord. in screen space	32
y_image	Vertex position Y-Coord. in screen space	32
z_image	Vertex position Z-Coord. in screen space	32

Scenario

First: Perform backface culling algorithm.
Second: Clipping Algorithm for only visible triangles.
Third: For only vertices in the view plane, perform perspective division and then multiply the result by the view port matrix to get the final vertices in screen space.

Internal Blocks

Backface culling: Perform Backface culling algorithm to avoid overloading of invisible triangles.
Scenario : First get the normal of triangle that represented by three vertices , but note that input is one vertex so you have to wait till all vertices transmitted then perform normal. Also, the vertex data is the modified positional data.
Second pass each vertex coordinates and the normal to perform backface culling algorithm.
Third, the output signal “cull_triangle” indicates if this tringle is visible or not. If not visible, then remove its corresponding data otherwise remain.

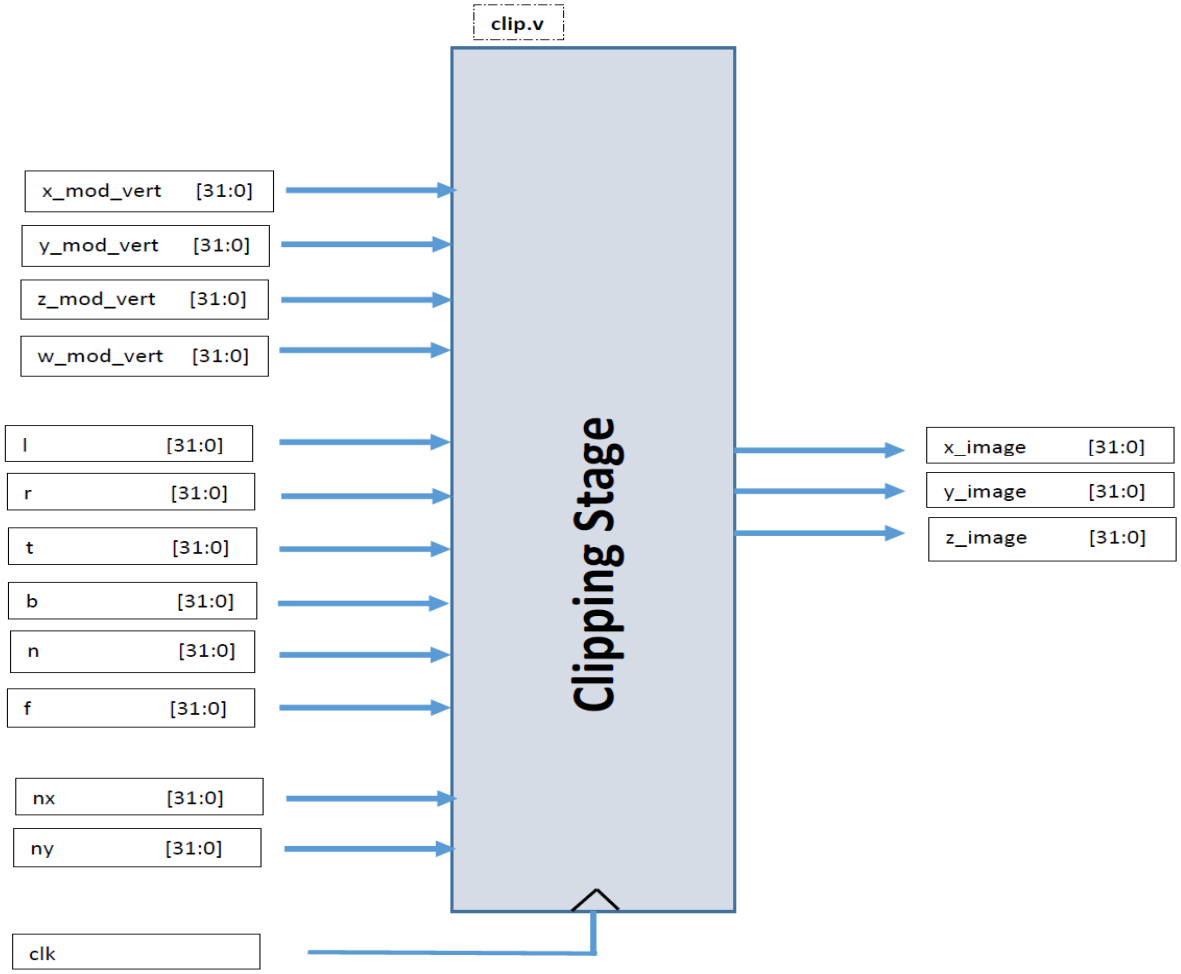
Clipping Algorithm: Note, after performing culling we will only get the vertices of only visible triangles. Only these vertices will be passed to this block to perform Cohen-Sutherland Algorithm in only 3D space. The output signals of this stage indicate the modified or same “for non- clipped vertices” vertices after clipping. Also, accept and reject role are declared in Control unit block.

Control: This unit gets their inputs from other inside block**.**
"cull_triangle" signal: this will effect if this triangle will be removed **"rv_cull"** or remained **"rm_cull"**.
"accept" signal : For lines inside view plane, these lines will remain and their vertices not clipped and pass directly their data without perform clipping algorithm through x_clip,y_clip,z_clip to the next stage "NDC". For lines that will be clipped ,**"up_clip"** signal will be enable to update their new clipped vertices by passing them through output data x_clip,y_clip,z_clip.
"reject" signal : For lines that will be totally rejected , then **"rv_clip"** remove clip will be enabled to remove their data and not pass them to next stage "NDC" The same concept for part of lines that will be rejected.

Matrix View-port Construction: Construct the view-port matrix.

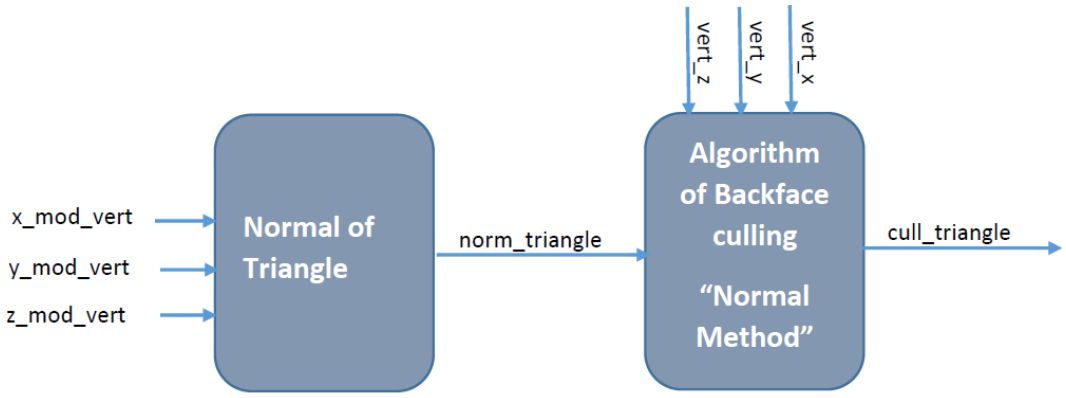
NDC "Normalized Device Coordinate" : is the perspective division to get the real positional vertices.

Block Diagram

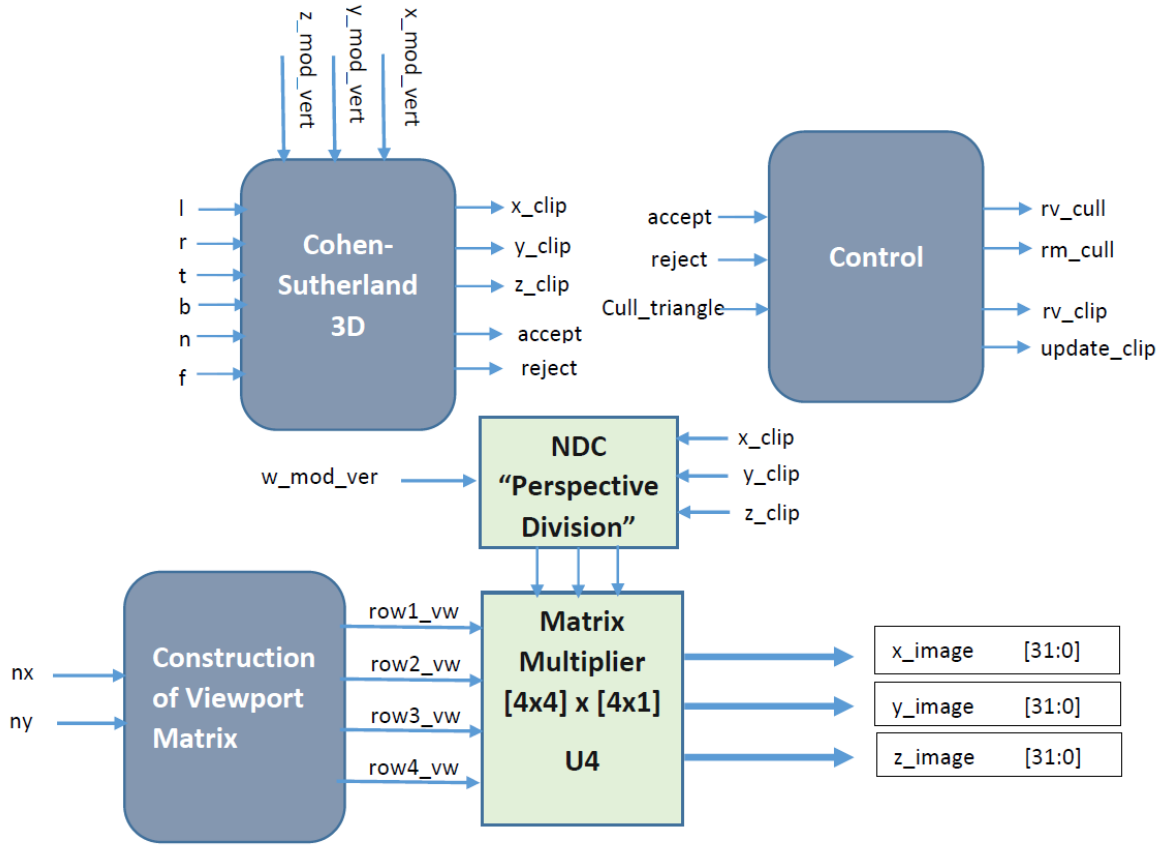


Interior view of clipping stage

Backface culling



Clipping Algorithm



Data Stored in Vertex Memory

Word-Name	Word-Number	Word-Size
Control	0	32
Vertex_data_X-Coord	1	32
Vertex_data_Y-Coord	2	32

Word-Name	Word-Number	Word-Size
Vertex_data_Z-Coord	3	32
Vertex_U-Coord	4	32
Vertex_V-Coord	5	32
Camera_Position_X(Ex)	6	32
Camera_Position_Y(Ey)	7	32
Camera_Position_Z(Ez)	8	32
Camera_Direction_X(Gx)	9	32
Camera_Direction_Y(Gy)	10	32
Camera_Direction_Z(Gz)	11	32
Camera_Up_X(Tx)	12	32
Camera_Up_Y(Ty)	13	32
Camera_Up_Z(Tz)	14	32
World_Space_Left_Side(L)	15	32
World_Space_Right_Side(R)	16	32
World_Space_Top_Side(T)	17	32
World_Space_Bottom_Side(B)	18	32
World_Space_Near_Side(N)	19	32
World_Space_Far_Side(F)	20	32
W	21	32
Screen_Width(Nx)	22	32
Screen_Height(Ny)	23	32
Texture_S	24	32
Texture_T	25	32
Texture_R	26	32
Texture_Q	27	32
Vertex_Normal_X-Coord	28	32
Vertex_Normal_Y-Coord	29	32
Vertex_Normal_Z-Coord	30	32
Vertex_Color	31	32

Control Register

Field Name	Bits-Number	Description
en_t	0	Enable Translate Block
tx	1	Translate According to X-Coord
ty	2	Translate According to Y-Coord

Field Name	Bits-Number	Description
tz	3	Translate According to Z-Coord
en_r	4	Enable Rotation Block
rx	5	Rotate according to X-Coord
ry	6	Rotate according to Y-Coord
rz	7	Rotate according to Z-Coord
en_s	8	Enable Scaling Block
sx	9	Scaling in X-Coord
sy	10	Scaling in Y-Coord
sz	11	Scaling in Z-Coord
en_u1	12	Enable of Matrix Multiplier between Translate and Rotate Blocks
en_u2	13	Enable of Matrix Multiplier between Scale and Output of Matrix Multiplier U1.
p_ortho	14	For Orthographic Projection
p_persp	15	For Perspective Projection
en_norm	16	Enable Normal Block
en_text	17	Enable Texture Block
cw	18	Rotate Clock Wise
ccw	19	Rotate Counter Clock Wise

OpenGL Supported Commands

Supported Commands	Suggested Address for Allocation
Void MatrixMode (enum mode)	0X 0000
Void LoadMatrix{xf}(T m[16])	0X 0004
Void MultMatrix{xf}(T m[16])	0X 0008

Supported Commands	Suggested Address for Allocation
Void LoadIdentity (void)	0X 000C
Void Active Texture (enum texture)	0X 0010
Void Enable (enum target)	0X 0014
Void Disable (enum target)	0X 0018
Void Translate{xf} (Tx,Ty,Tz)	0X 001C
Void Scale{xf} (Tx,Ty,Tz)	0X 0020
Void Rotate{xf} (Tθ,Tx,Ty,Tz)	0X 0024
Void frunstum{xf} (Tl,Tr,Tb,Tt,Tn,Tf)	0X 0028
Void Ortho{xf} (Tl,Tr,Tb,Tt,Tn,Tf)	0X 002C
Void Translate{xf} (Tx,Ty,Tz)	0X 0030
Void ClipPlane fxfg(enum p,Const T eqn[4])	0X 0034
glenable (GL_Clip_Plane i)	0X 0038
gldisable (GL_Clip_Plane i)	0X 003C
Void viewport(int x,int y,size i w,size i h)	0X 0040

Rasterization

- The rasterization has two main functions that are listed as follow:
    1. Determine which squares of an integer grid window are occupied (pixels).
    2. Assigning color, depth and texture coordinates of each pixel.
- The rasterization can be done on any shape, but we will be concerned only the supported shapes in OpenGL which are three shapes that are point, line, and triangle.
- The following figure shows a functional block diagram of the rasterizer block, this block diagram will be enhanced in the future and the HW block diagram will be derived from it.

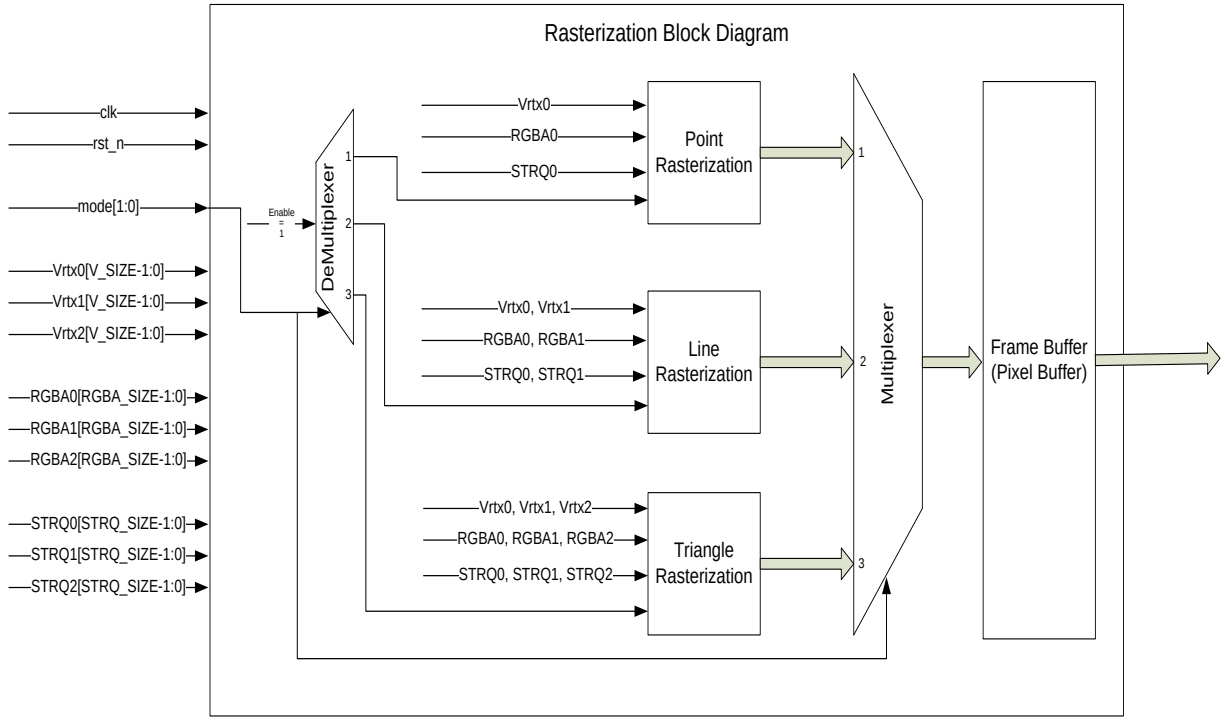


Figure 4: Rasterization Functional Block Diagram

- The rasterizer block shown in Figure 1 has three ports for each of the vertices, RGBA (color and luminance) and STRQ (texture coordinates). Depending on the mode (1: Point, 2: Line, 3: Triangle) it will select the first vertices that will form the required shape selected by mode.
- The rasterization of each mode requires some associated parameters with it. These parameters will be either set by the software and it will be used every time with its corresponding shape, or the parameters will be fed to the rasterizer block with each new input. Further discussion is needed here.
- The parameters needed by the point, the line and the triangle with their OpenGL commands are shown in Table 1

Table 1: Point Rasterization Parameters

Parameter Name	OpenGL Floating Command	OpenGL Fixed Command	*target*	*pname*	*params*	Value
Size	void **PointSize**(float *size*)	void **PointSizex**(fixed *size*)	--	--	--	>0
Distance Attenuation Function Coefficients {*a, b, c*}	void **PointParameter{xf}**(enum *pname*, T *param*)	**void PointParameter{xf}v**(enum *pname*, const T *params*)	--	POINT_DISTANCE_ATTENUATION	a, b and c	??

Parameter Name	OpenGL Floating Command	OpenGL Fixed Command	*target*	*pname*	*params*	Value
Minimum Point Size	void **PointParameter{xf}**(enum *pname*, T *param*)	**void PointParameter{xf}v**(enum *pname*, const T *params*)	--	POINT_SIZE_MIN	{Upper Bound, Lower Bound}	>0
Maximum Point Size	void **PointParameter{xf}**(enum *pname*, T *param*)	**void PointParameter{xf}v**(enum *pname*, const T *params*)	--	POINT_SIZE_MAX	{Upper Bound, Lower Bound}	>0
Point Fade Threshold	void **PointParameter{xf}**(enum *pname*, T *param*)	**void PointParameter{xf}v**(enum *pname*, const T *params*)	--	POINT_FADE_THRESHOLD_SIZE	Points To The Point Fade Threshold	>0
Point Antialiasing	void **Enable**(enum *target*) or void **Disable**(enum *target*)	void **Enable**(enum *target*) or void **Disable**(enum *target*)	POINT_SMOOTH	--	--	{Enable, Disable}
Point Sprites	void **Enable**(enum *target*) or void **Disable**(enum *target*)	void **Enable**(enum *target*) or void **Disable**(enum *target*)	POINT_SPRITE_OES	--	--	{Enable, Disable}
Point Sprite Texture Coordinate Replacement Mode	void **TexEnv{ixf}(** enum *target*, enum *pname*, T *param*)	void **TexEnv{ixf}v(** enum *target*, enum *pname*, T *param*)	POINT_SPRITE_OES	COORD_REPLACE_OES	{FALSE, TRUE}	{FALSE, TRUE}

- There are also common parameters needed by the three shapes. These parameters are mainly for enabling/disabling the multisampling, and they are shown in Table 2.

Table 2: Common Rasterization parameters

Parameter Name	OpenGL Floating Command	OpenGL Fixed Command	*target*	*pname*	*params*	Value
Multisampling	void **Enable**(enum *target*) or	void **Enable**(enum *target*) or	MULTISAMPLE	--	--	Enable, Disable}

Parameter Name	OpenGL Floating Command	OpenGL Fixed Command	*target*	*pname*	*params*	Value
Sample Buffers	void **Disable**(enum *target*)	void **Disable**(enum *target*)				
			--	SAMPLE BUFFERS	--	Integer (1: Eanbled, else: Disabled

- The algorithms for rasterizing points, lines, and triangles are beyond this document.

Texturing

Introduction

This specification document aims to describe the texturing in OpenGL ES 1.1.12. The document will illustrate the supported feature set, along with the unsupported features and the simplifications assumed in the texturing process

Supported Commands

According to the OpenGL ES 1.1.12 standard, the GPU 2015 texturing system will support the following commands:

glEnable(GL_TEXTURE_2D)

Enables the texturing in the texture-mapping unit, if user did not issue such command, no texturing will take place

glTexImage2D

Is used to specify a texture image, the image data is located in the host memory

glCopyTexImage2D

Acts same as the previous command, except that the image is located in the framebuffer rather than the host memory

glTexParameter

Specifies how the texture array is treated when specified or changed, and when applied to a fragment, all the parameter values are supported except for the "GENERATE_MIPMAP" is not supported and must be set to "FALSE"

glTexEnv

Sets parameters of the texture environment that specifies how texture values are interpreted when texturing a fragment

Unsupported Commands/Features

glTexSubImage2D

glCopyTexSubImage2D

glCompressedTexImage2D

glCompressedTexSubImage2D

glBindTexture

glDeleteTextures

glGenTextures

Texturing Block Diagram

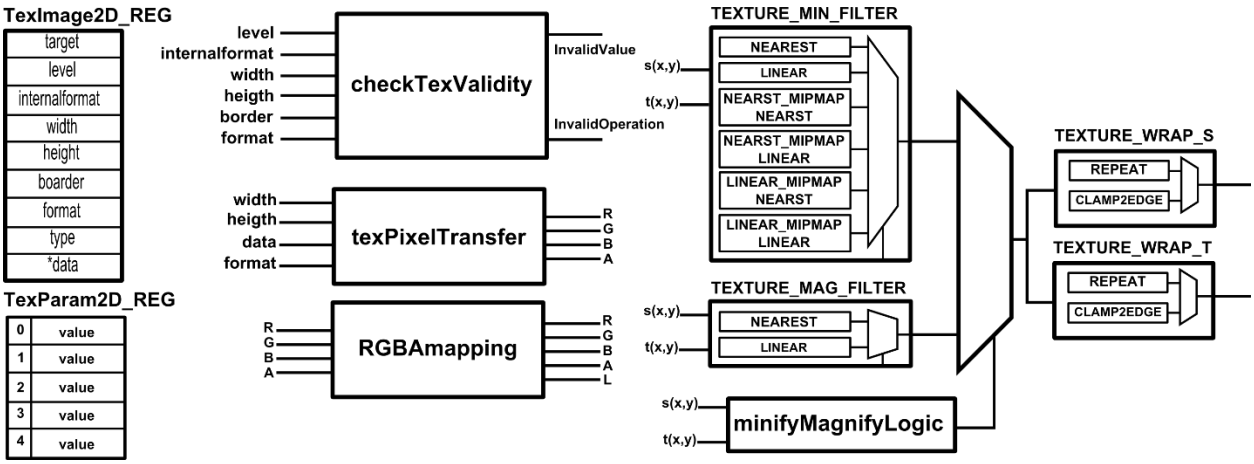


Figure 5. Texturing system level diagram

Block Description

TexImage2D_REG

It is a register holding the arguments passed to the "*glTexImage2D*" command, the saved values are directly accessed by different blocks in the system.

Description of such register file can be summarized in Table 3. For instance, the size and location of the bits can be changed according to the system resources and the designer point of view.

Table 3: TexImage2D_REG description

Address	Name	Size (bits)
0	target	32
1	level	3
2	internalformat	3
3	width	32
4	height	32
5	boarder	32
6	format	3
7	type	3
8	*data	32

TexParam2D_REG

Register holding the texture parameter set via the "*glTexParamter*". Table 4 details the parameter address and the corresponding value for each. For instance, the size and location of the bits can be changed according to the system resources and the designer point of view.

Table 4: TexParam2D_REG description

Address	Name	Size (bits)	Value
0	TEXTURE_WRAP_S	1	0: REPEAT 1: CLAMP_TO_EDGE
1	TEXTURE_WRAP_T	1	0: REPEAT 1: CLAMP_TO_EDGE
2	TEXTURE_MIN_FILTER	3	0: NEARST 1: LINEAR 2: NEARST_MIPMAP_NEARST 3: NEARST_MIPMAP_LINEAR 4: LINEAR_MIPMAP_NEARST 5: LINEAR_MIPMAP_LINEAR
3	TEXTURE_MAG_FILTER	1	0: NEARST 1: LINEAR
4	GENERATE_MIPMAP	0	0

CheckTexValidity

This block is responsible for checking the correctness of the parameter passed to the "glTexImage2D" command, implementation of the two output signals are detailed in the standard in the texturing section 3.7.1, section 3.7.2 and section 3.7.12

TexPixelTransfer

Responsible for expanding the image pixels to RGBA format, details of handling such processing can be found in the OpenGL ES standard in section 3.6.2

As per the architecture design, the output color data is a signed 32 bit fixed point bus.

RGBAmapping

Maps the expanded RGBA output from the previous block to internal texture components.

As per the architecture design, the output color data is a signed 32 bit fixed point bus.

TEX_MIN_FILTER

It is the texture minification algorithms block, either minification algorithm is selected based upon the configuration saved in the TexParam2D_REG

TEX_MAG_FILTER

It is the texture magnification algorithms block, either magnification algorithm is selected based upon the configuration saved in the TexParam2D_REG

minifyMagnifyLogic

it's the logic circuit which decides either to magnify or minify the texture to fit in specified geometry, exact decision criteria is detailed in sections 3.7.7 and 3.7.8

TEX_WRAP_S

Wraps the 's' coordinate based upon the configuration stored in the TexParam2D_REG

TEX_WRAP_T

Wraps the 't' coordinate based upon the configuration stored in the TexParam2D_REG