GTE & Advanced Graphics

Organization of this talk

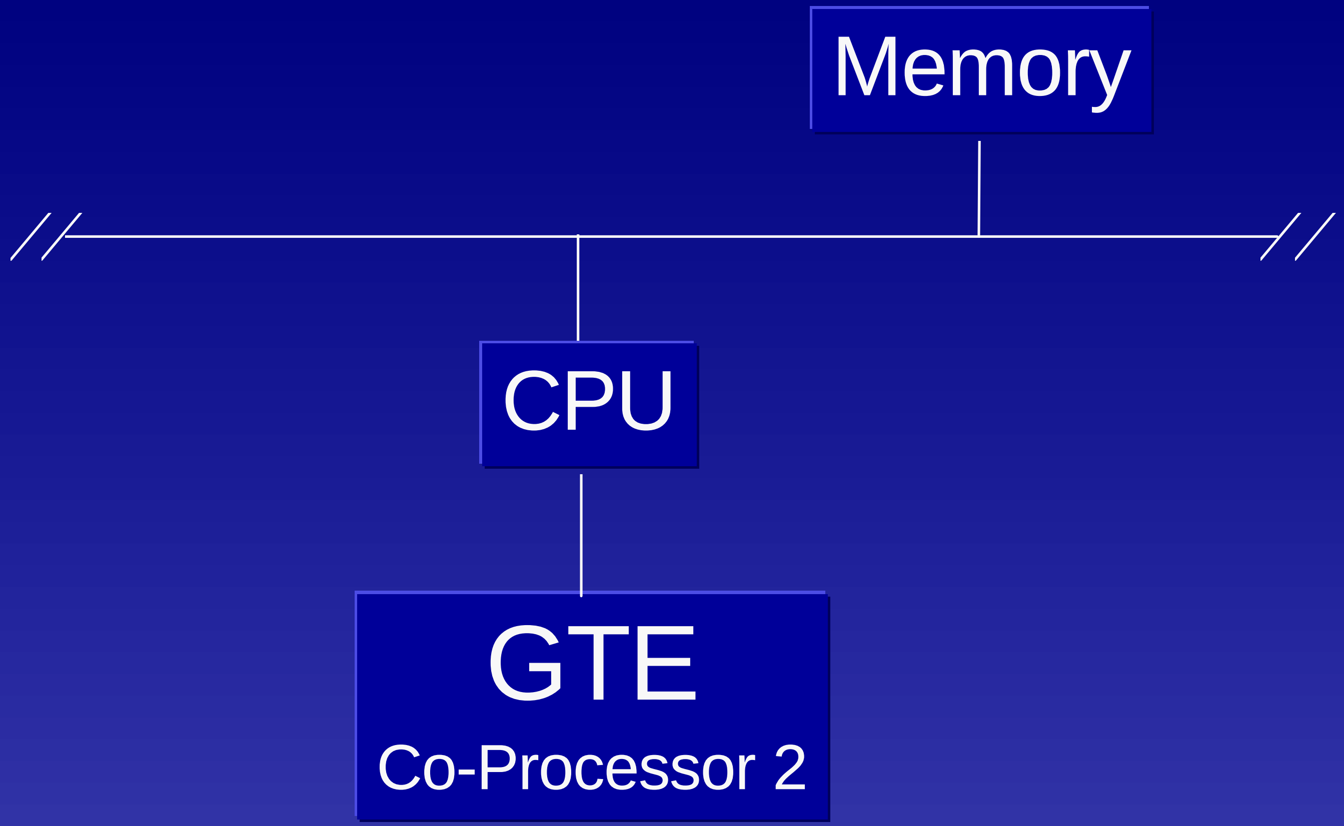
- ✓ Part 1: description of GTE hardware
- ✓ Part 2: the new, improved DMPSX
- ✓ Part 3: revisit some old favorites
- ✓ Q&A

Part 1:
Everything there is to know about the GTE Hardware

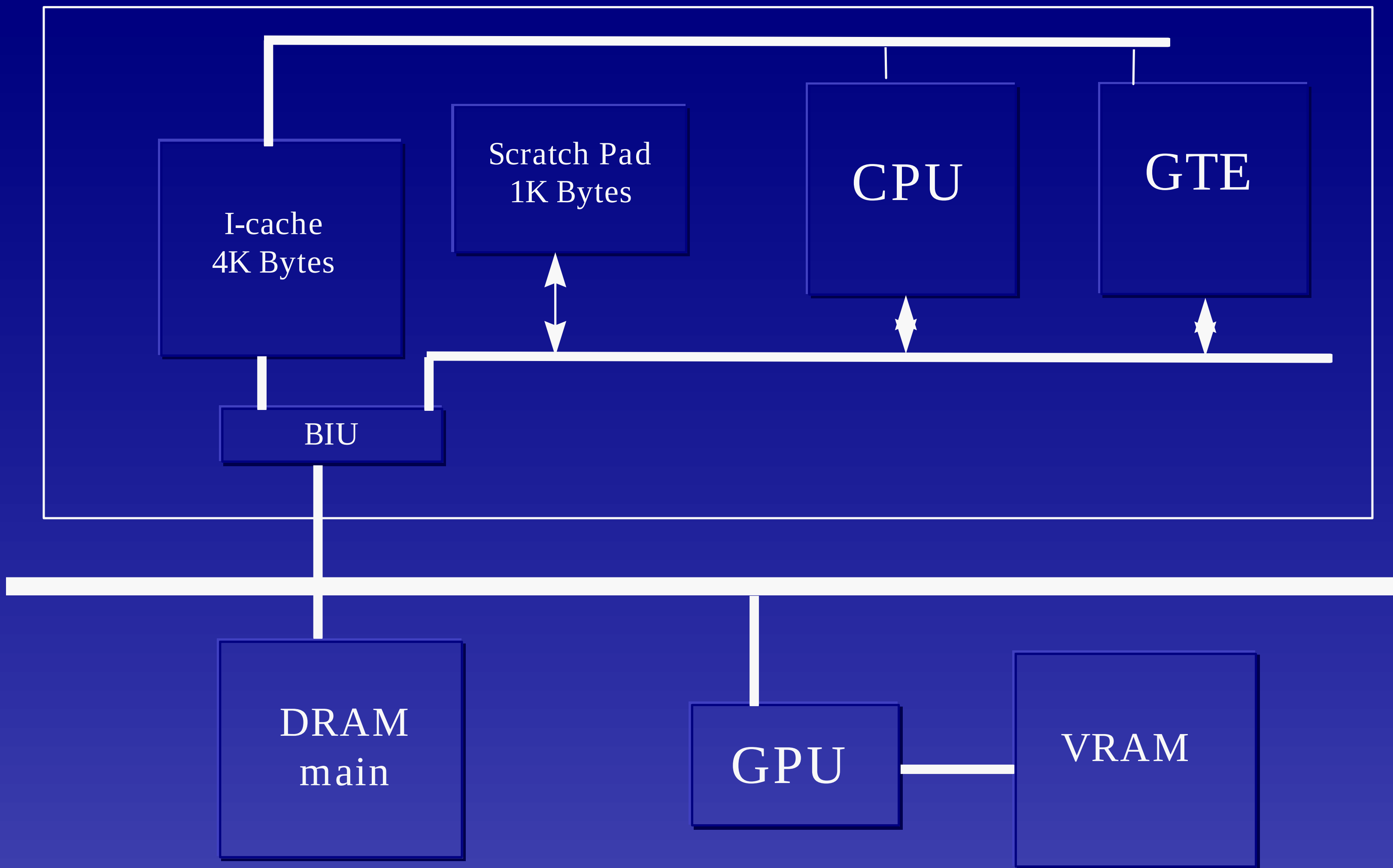
MIPS R3000 and COPROCESSOR UNITS

- ✓ MIPS architecture defines four *coprocessor units*, Coprocessor 0-3

CPU+GTE

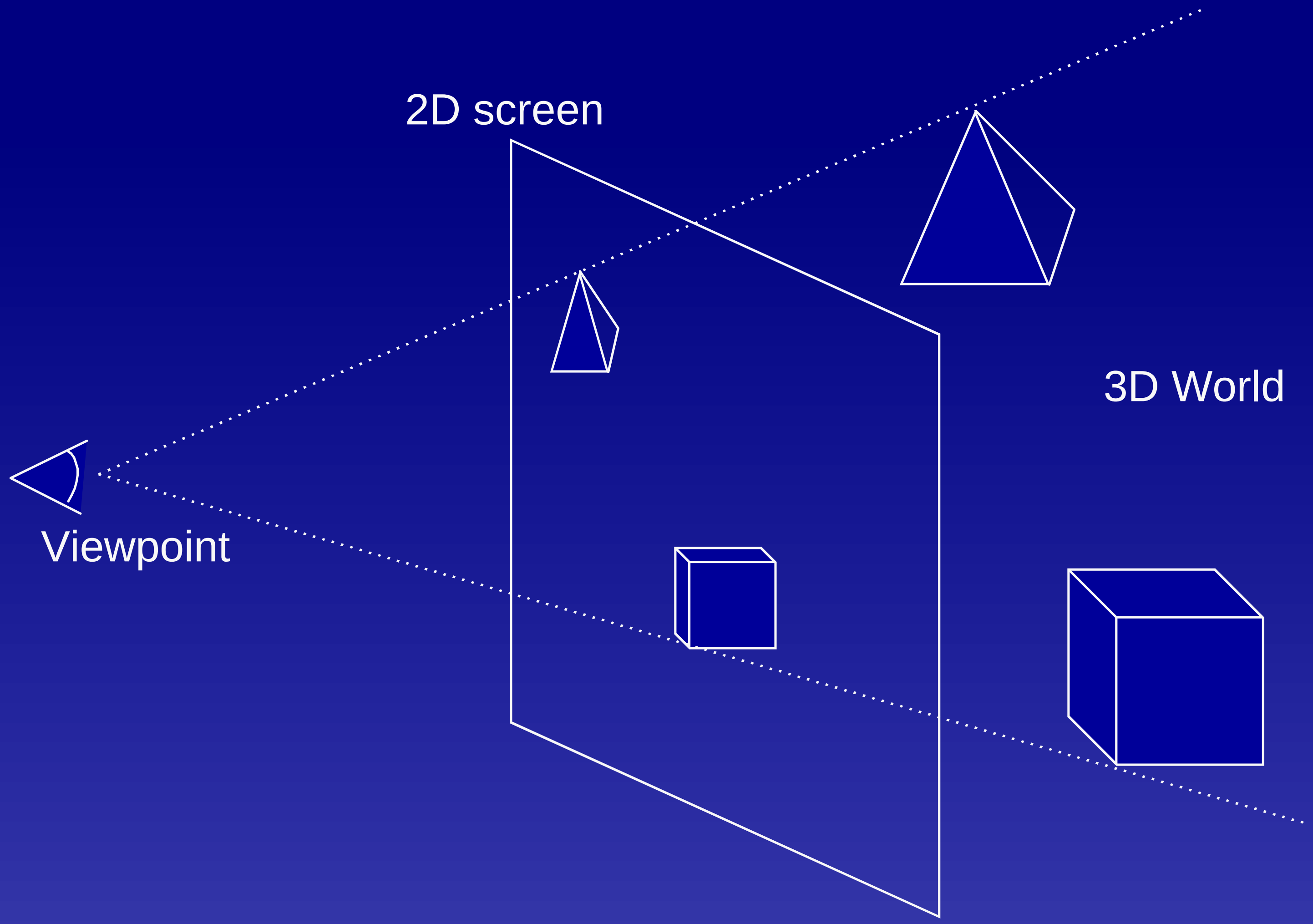


CPU Block Diagram



What exactly is the GTE ?

3D->2D



...What exactly is the GTE ?

- ✓ GTE is a vector / matrix high speed geometric processor with its own multiplier, accumulator and divider, implemented as "coprocessor 2" under the MIPS architecture specification.
- ✓ The data format supported by GTE consists of fixed decimal(fractional) real numbers.

GTE features

- ✓ High speed matrix calculations
- ✓ High speed coordinate transformations
- ✓ High speed perspective projections
- ✓ High speed lighting calculations

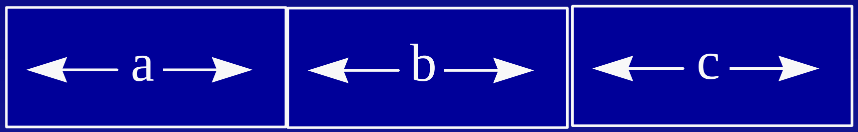
The Mathematics Behind the GTE

The mathematics behind the GTE

- ✓ Number system representation
- ✓ The GTE calculations
  - coordinate calculations
  - light source calculations

Fixed point representations

Fixed point bit arrangement



a: Signed bit

b: Integer bits

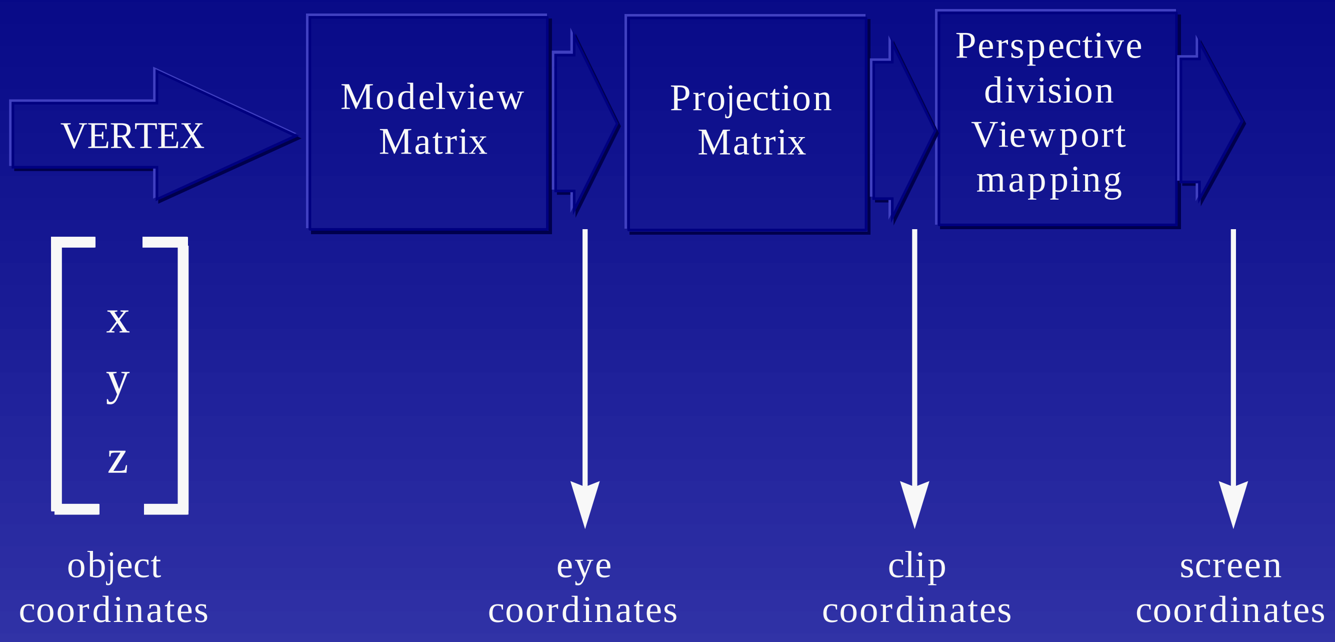
c: Decimal bits

Some existing representations...

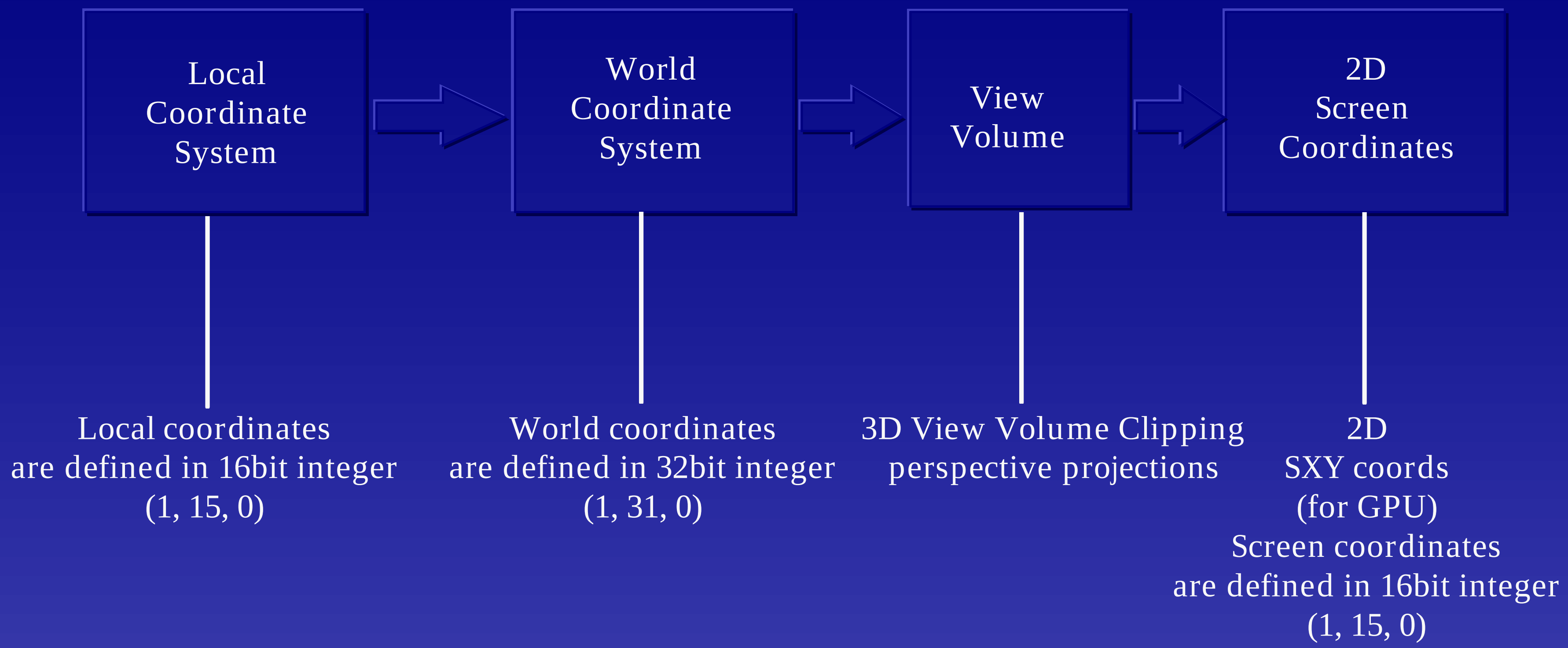
- Rotational matrix [Rij] (1, 3, 12)
- Translating vector (TRX, TRY, TRZ) (1, 31, 0)
- Local light matrix [Lij] (1, 3, 12)
- Local color matrix [L (R, G, B)ij] (1, 3, 12)
- Back color (RBK, GBK, BBK) (0, 8, 0)
- Far color (RFC, GFC, BFC) (0, 8, 0)

A sample geometry pipeline...

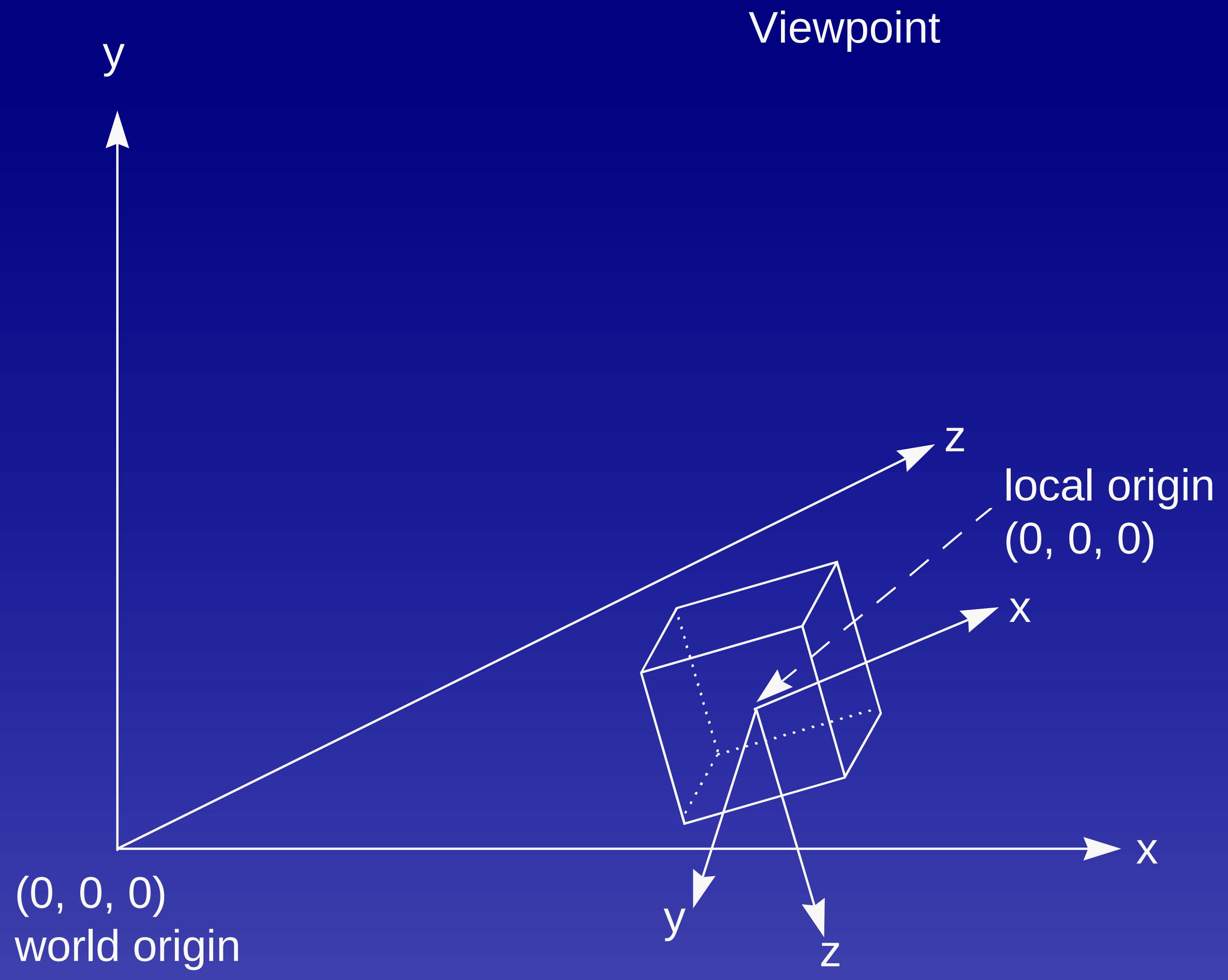
...for vertex transformation...



GTE implementation



GTE coordinate system



GTE calculations

rot,trans, matrix

World to Screen translation

Screen Vertex

1 $$\begin{bmatrix} SSXn \\ SSYn \\ SZn \end{bmatrix} = \mathbf{R2} \begin{bmatrix} WX \\ WY \\ WZ \end{bmatrix} + \mathbf{WT}$$

2 $$\begin{bmatrix} WX \\ WY \\ WZ \end{bmatrix} = \mathbf{R1} \begin{bmatrix} LX \\ LY \\ LZ \end{bmatrix} + \mathbf{LT}$$

Local Coordinates

rot,trans matrix

Local to World translation

aggregate local-to-screen translation vector

Screen Vertex

1’ $$\begin{bmatrix} SSXn \\ SSYn \\ SZn \end{bmatrix} = \mathbf{R2}\ \mathbf{R1} \begin{bmatrix} LX \\ LY \\ LZ \end{bmatrix} + \mathbf{T}$$

Local Coordinates

aggregate local-to-screen transformation matrix

GTE calculations...

World Level

$$\begin{bmatrix} WS11 & WS12 & WS13 \\ WS21 & WS22 & WS23 \\ WS31 & WS32 & WS33 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & Cx & Sx \\ 0 & -Sx & Cx \end{bmatrix} \begin{bmatrix} Cy & 0 & Sy \\ 0 & 1 & 0 \\ -Sy & 0 & Cy \end{bmatrix} \begin{bmatrix} Cz & Sz & 0 \\ -Sz & Cz & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Transformation
Matrix R2
World -to- Screen

X-rot

Y-rot

Z-rot

$C = \cos(\alpha)$

$S = \sin(\alpha)$

GTE calculations...

Object Level

$$
\begin{bmatrix} LW11 & LW12 & LW13 \\ LW21 & LW22 & LW23 \\ LW31 & LW32 & LW33 \end{bmatrix}
=
\begin{bmatrix} 1 & 0 & 0 \\ 0 & Cx & Sx \\ 0 & -Sx & Cx \end{bmatrix}
\begin{bmatrix} Cy & 0 & Sy \\ 0 & 1 & 0 \\ -Sy & 0 & Cy \end{bmatrix}
\begin{bmatrix} Cz & Sz & 0 \\ -Sz & Cz & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$

Transformation Matrix R1 Local -to- World

X-rot

Y-rot

Z-rot

$C = \cos(\alpha)$

$S = \sin(\alpha)$

GTE calculations...

Object Level

$$
\begin{bmatrix} R11 & R12 & R13 \\ R21 & R22 & R23 \\ R31 & R32 & R33 \end{bmatrix}
=
\begin{bmatrix} WS11 & WS12 & WS13 \\ WS21 & WS22 & WS23 \\ WS31 & WS32 & WS33 \end{bmatrix}
\begin{bmatrix} LW11 & LW12 & LW13 \\ LW21 & LW22 & LW23 \\ LW31 & LW32 & LW33 \end{bmatrix}
$$

aggregate
local-to-screen
transformation
matrix
R2 R1

World -to- Screen
Transformation
Matrix R2

Local -to- World
Transformation
Matrix R1

GTE calculations...

Object Level

$$\begin{bmatrix} TrX \\ TrY \\ TrZ \end{bmatrix} = \begin{bmatrix} WS11 & WS12 & WS13 \\ WS21 & WS22 & WS23 \\ WS31 & WS32 & WS33 \end{bmatrix} \begin{bmatrix} TLX \\ TLY \\ TLZ \end{bmatrix} + \begin{bmatrix} TWX \\ TWY \\ TWZ \end{bmatrix}$$

T
aggregate local-to-screen translation vector

World -to- Screen Transformation Matrix R2

LT
Local to World translation

WT
World to Screen translation

GTE calculations...

Polygon Level

$$\begin{bmatrix} SSXn \\ SSYn \\ SZn \end{bmatrix} = \begin{bmatrix} R11 & R12 & R13 \\ R21 & R22 & R23 \\ R31 & R32 & R33 \end{bmatrix} \begin{bmatrix} VXn \\ VYn \\ VZn \end{bmatrix} + \begin{bmatrix} TrX \\ TrY \\ TrZ \end{bmatrix}$$

Screen Vertex

R2 R1

aggregate
local-to-screen
transformation
matrix

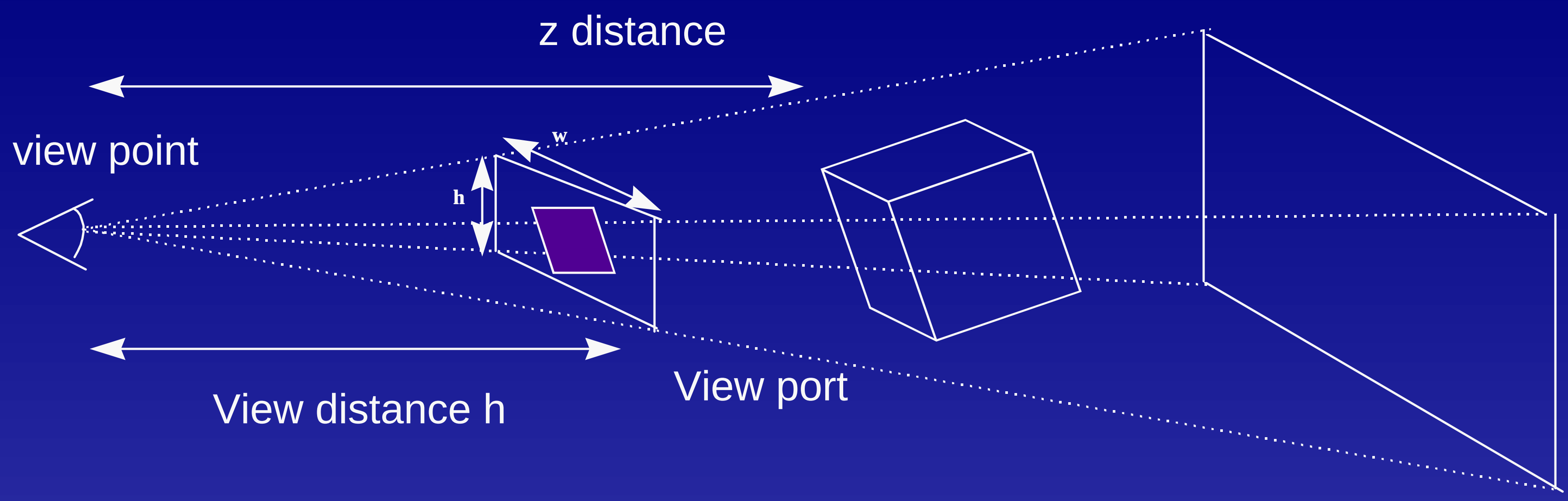
local Vertex

T

aggregate
local-to-screen
translation
vector

GTE calculations...

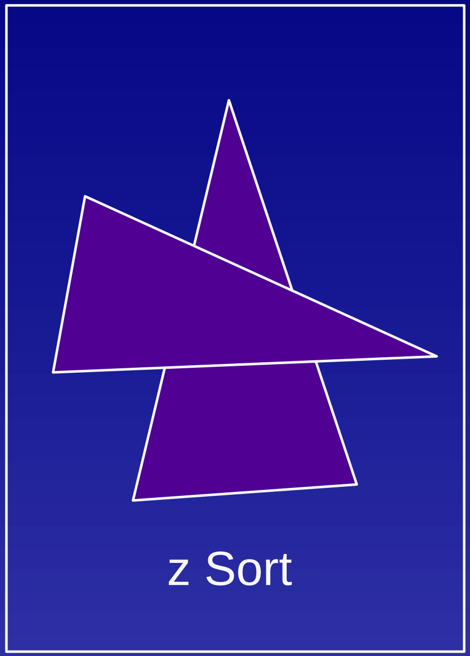
perspective calculation



SXn = OFX + SSXn * (h/SZn);
SYn = OFY + SSYn * (h/SZn);

GTE calculations...

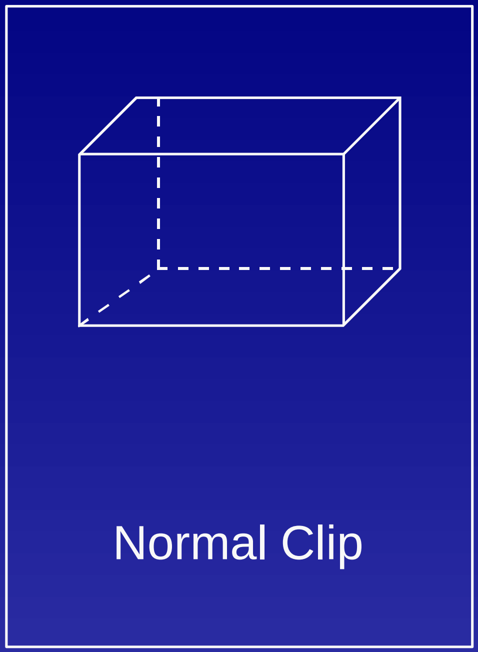
- hidden surface removal



OTZ = SZ0 *(1/3) + SZ1*(1/3) + SZ2*(1/3);

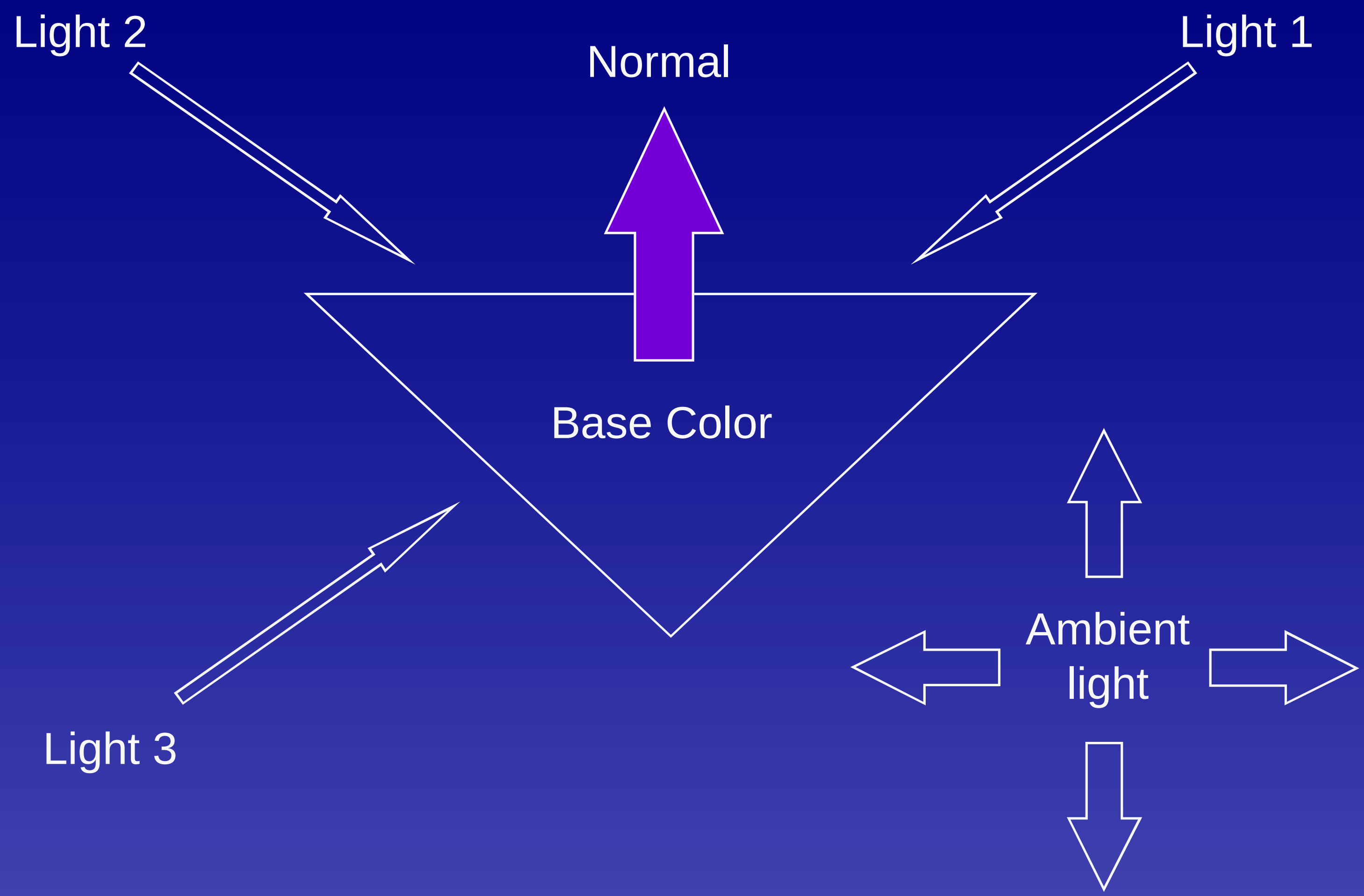
GTE calculations...

- hidden surface removal...

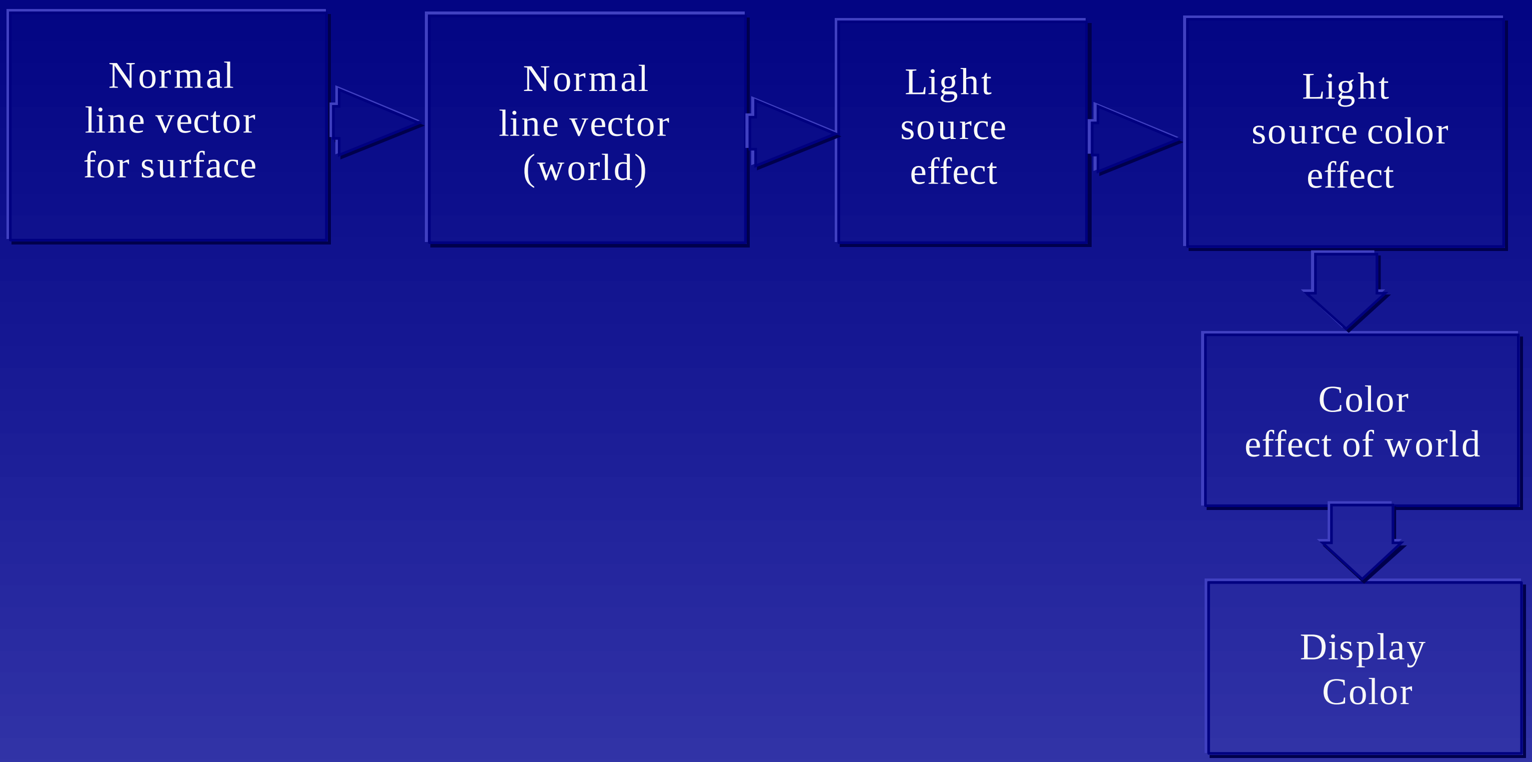


OPZ = (SX1-SX0)(SY2-SY0) - (SX2-SX0)(SY1-SY0);
= (SX0SY1+SX1SY2+SX2SY0)-(SX0SY2+SX1SY0+SX2SY1);

Light source calculations...



GTE Light Source Calculation



GTE light source calculations...

Object Level

$$\begin{bmatrix} L11 & L12 & L13 \\ L21 & L22 & L23 \\ L31 & L32 & L33 \end{bmatrix} = \begin{bmatrix} L1X & L1Y & L1Z \\ L2X & L2Y & L2Z \\ L3X & L3Y & L3Z \end{bmatrix} \begin{bmatrix} LW11 & LW12 & LW13 \\ LW21 & LW22 & LW23 \\ LW31 & LW32 & LW33 \end{bmatrix}$$

Local Lights — Light vectors — Transformation Matrix R1 Local -to- World

GTE light source calculations...

Polygon Level

$$\begin{bmatrix} L1 \\ L2 \\ L3 \end{bmatrix} = \begin{bmatrix} L11 & L12 & L13 \\ L21 & L22 & L23 \\ L31 & L32 & L33 \end{bmatrix} \begin{bmatrix} NX \\ NY \\ NZ \end{bmatrix}$$

Local Lights

Local Normal Vector

GTE light source calculations...

Polygon Level

$$\begin{bmatrix} \mathrm{RLT} \\ \mathrm{GLT} \\ \mathrm{BLT} \end{bmatrix} = \begin{bmatrix} \mathrm{LR1} & \mathrm{LR2} & \mathrm{LR3} \\ \mathrm{LG1} & \mathrm{LG2} & \mathrm{LG3} \\ \mathrm{LB1} & \mathrm{LB2} & \mathrm{LB3} \end{bmatrix} \boxed{\begin{bmatrix} \mathrm{L1} \\ \mathrm{L2} \\ \mathrm{L3} \end{bmatrix} \text{ or } \begin{bmatrix} \mathrm{L1}^n \\ \mathrm{L2}^n \\ \mathrm{L3}^n \end{bmatrix}} + \begin{bmatrix} \mathrm{RBK} \\ \mathrm{GBK} \\ \mathrm{BBK} \end{bmatrix}$$

Local Light color | Light Color Matrix | Light Source Effect | Ambient Light color

GTE light source calculations

Polygon Level

$$\begin{bmatrix} R0 \\ G0 \\ B0 \end{bmatrix} = (1-p) \begin{bmatrix} R \times RLT \\ G \times GLT \\ B \times BLT \end{bmatrix} + p \begin{bmatrix} RFC \\ GFC \\ BFC \end{bmatrix}$$

Screen Color

Polygon Color

Far Color

p=DQA*h/SZn +DQB

GTE Register Set

- ✓ The GTE has two sets of registers
- ✓ 32 control registers and 32 general (data) registers

GTE register set...

- ✓ general(data) registers

VX0	VY0	VZ0
VX1	VY1	VZ1
VX2	VY2	VZ2

Input vector-Vn(dreg0~5) **R/W**

R	G	B	cd

24bit Color Input+GPU code (dreg 6) **R/W**

OTZ

Average of Z-data(dreg7) 1,15,0 **R** OTZ Register

GTE register set...

- ✓ general(data) registers

IR0

Intermediate Register (dreg8) **R/W** (p)

IR1

Intermediate Register (dreg9) **R/W**

IR2

Intermediate Register (dreg10) **R/W**

IR3

Intermediate Register (dreg11) **R/W**

GTE register set...

- ✓ general(data) registers

SX0	SY0
SX1	SY1
SX2	SY2

FIFO

SX0SY0<-SX1SY1<-SX2SY2<-input

2D Vertex FIFO registers
(dreg12~14) **R/W**

SX2’	SY2’

Vertex FIFO register input
(dreg15) **W**

GTE register set...

- general(data) registers

SZx(0)

SZ0(1)

SZ1(2)

SZ2(3)

FIFO

SZx(0)<-SZ0(1)<-SZ1(2)-<SZ2(3)<-input

Screen-Z Vertex FIFO registers
(dreg16~19) **R/W**

R0, G0, B0, cd

R1, G1, B1, cd

R2, G2, B2, cd

C-FIFO

Color FIFO registers
(dreg20~22) **R/W**
RGB0-23;code:24-31

GTE register set...

- ✓ general(data) registers

$\boxed{\text{MAC-0}}$

MAC-0 Output (dreg24) **R**

$\boxed{\text{MAC-1}}$

MAC-1 Output (dreg25) **R/W**

$\boxed{\text{MAC-2}}$

MAC-2 Output (dreg26) **R/W**

$\boxed{\text{MAC-3}}$

MAC-3 Output (dreg27) **R/W**

GTE register set...

- ✓ general(data) registers

iRGB

15 bit color input (dreg28) **W**

oRGB

15 bit color output (dreg29) **R**

LZCS

Leading Zero Count Set

Leading Zero counter input (dreg30) **W**

LZCR

Leading Zero Count Read

Leading Zero counter output (dreg31) **R** 0-5;0:6-31;

GTE register set...

✓ control registers...

R11	R12	R13
R21	R22	R23
R31	R32	R33

Rotation Matrix (creg0~4) **R/W** (1,3,12)

TRX
TRY
TRZ

TranslationVector (creg5~7) **R/W** (1,31,0)

L11	L12	L13
L21	L22	L23
L31	L32	L33

Light Source direction vectorX3 (creg8~12) **R/W** (1,3,12)

GTE register set...

- ✓ control registers...

RBK
GBK
BBK

background color (creg13~15) **R/W** (1,19,12)

LR1	LR2	LR3
LG1	LG2	LG3
LB1	LB2	LB3

Light Source color vectorX3 (creg16~20) **R/W** (1,3,12)

RFC
GFC
BFC

far color (creg21~23) **R/W** (1,27,4)

GTE register set...

- ✓ control registers...

OFX
OFY

Screen Offset X&Y (creg24~25)**R/W** (1,15,16)

H

Screen Position (creg26) **R/W** (0,16,0)

DQA

Depth parameter A(coefficient) (creg27) **R/W** (1,7,8)

DQB

Depth parameter B(offset) (creg28) **R/W** (1,7,24)

GTE register set...

- ✓ control registers...

ZSF3
ZSF4

Z-averaging scale factors (creg29~30) **R/W** (1,3,12)

$$ZSF3=1/3*2^{10\sim 14}/2^{16}$$

$$ZSF4=1/4*2^{10\sim 14}/2^{16}$$

GTE register set...

✓ control registers...

FLAG

FLAG register (creg31) **R**

only care for bits 12~31

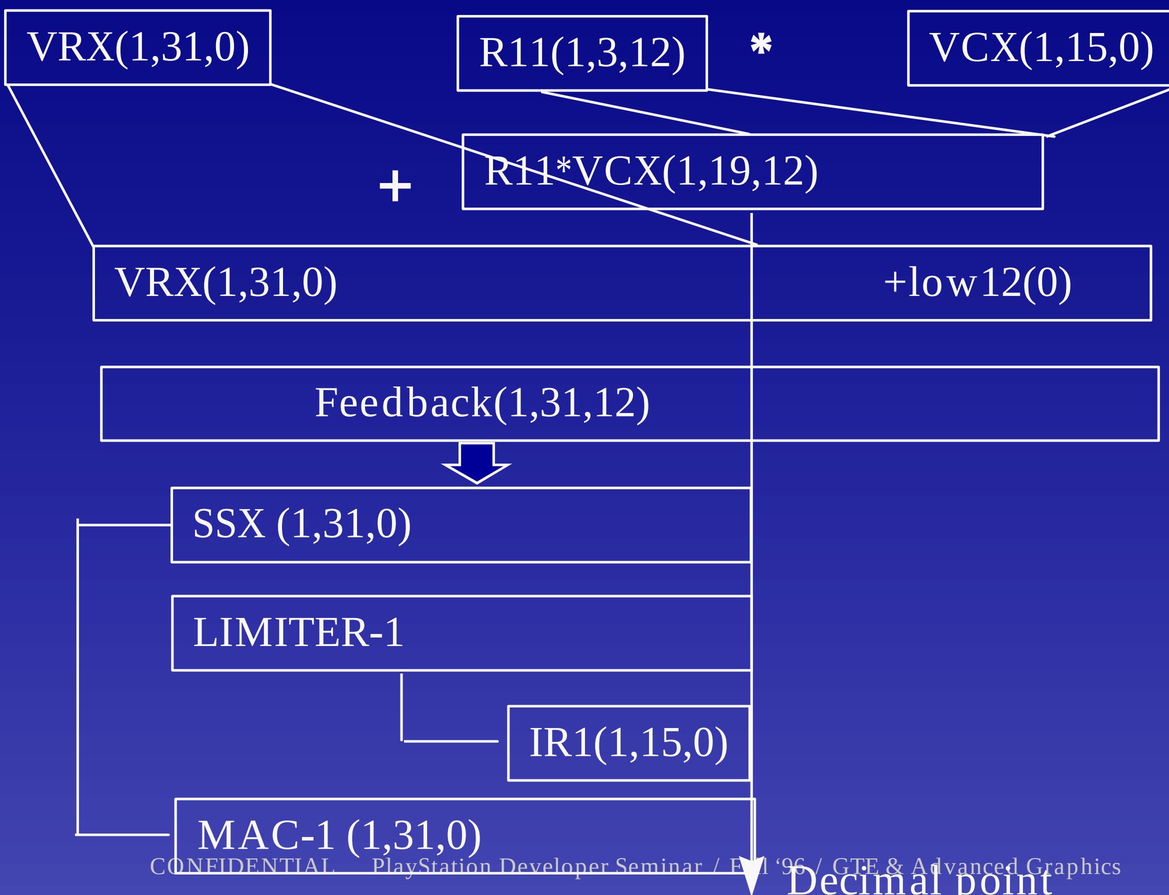
GTE Calculation Format

RTPS&RTPT&MVMVA(F3)

F3
MVMVA

SSXn = **TrX**+R11*VXn+R12*VYn+R13*VZn;
SSYn = **TrY**+R21*VXn+R22*VYn+R23*VZn;
SZn = TrZ+R31*VXn+R32*VYn+R33*VZn;

SSXn = **TrX**+R11*VXn+R12*VYn+R13*VZn;

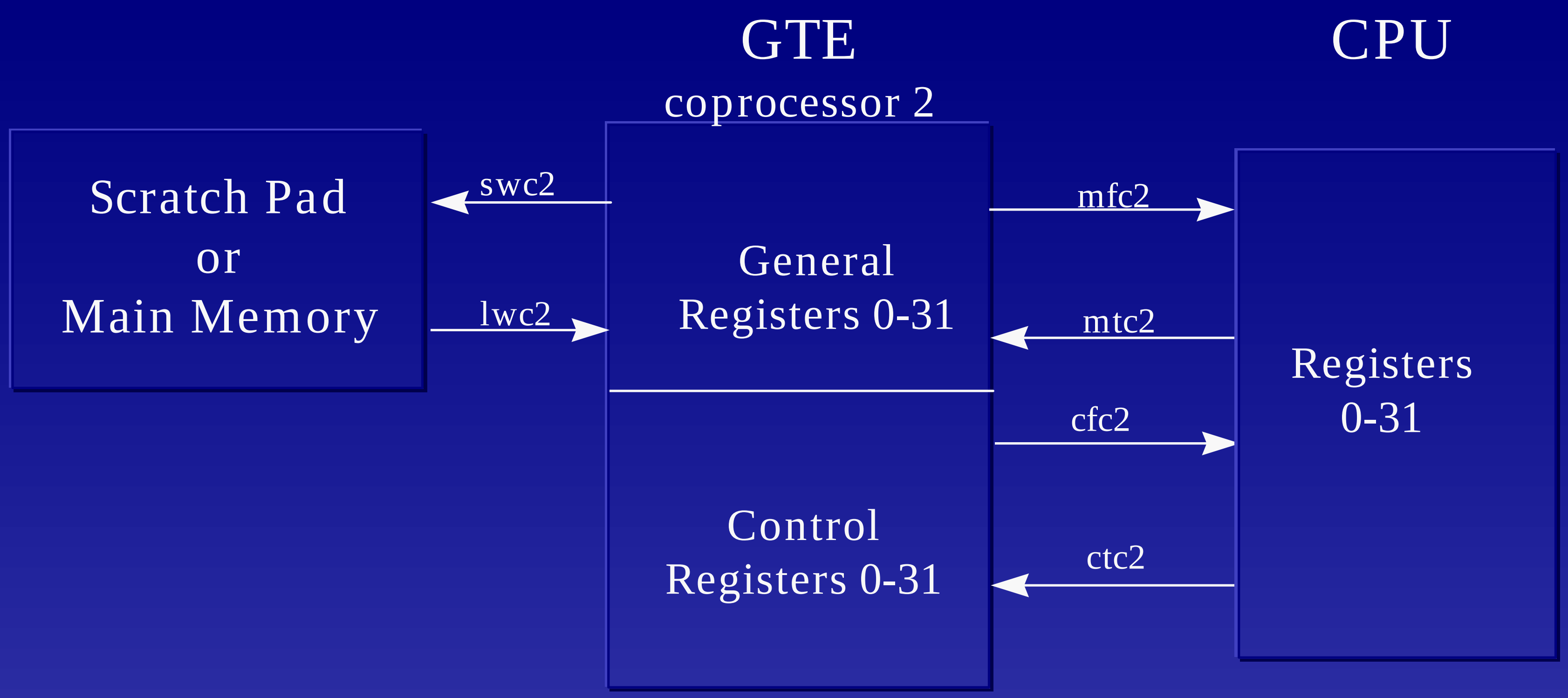


limiters?

- ✓ please refer to GTE Command Reference ver 1.0 page 2

Programming the GTE

GTE access instructions



Existing GTE command set

✓ coordinate calculations

Rot Trans Pers + Depth Calc	**F1,2** **(x3)** RTPS (RTPT)	SSXn = **TrX**+R11*VXn+R12*VYn+R13*VZn; SSYn = **TrY**+R21*VXn+R22*VYn+R23*VZn; SZn = TrZ+R31*VXn+R32*VYn+R33*VZn; SXn = OFX + SXn*(h / SZn); SYn = OFY + SYn*(h / SZn); p = DQB+DQA*(h / SZn);
Rot Trans	**F3** MVMVA	SSXn = **TrX**+R11*VXn+R12*VYn+R13*VZn; SSYn = **TrY**+R21*VXn+R22*VYn+R23*VZn; SZn = TrZ+R31*VXn+R32*VYn+R33*VZn;

Existing GTE command set…

✓ lighting calculations

Normal Color + Depth Cue	**F14,15** **(x3)** NCDS (NCDT)	L1 = L11*NXn+L12*NYn+L13*NZn; L2 = L21*NXn+L22*NYn+L23*NZn; L3 = L31*NXn+L32*NYn+L33*NZn; RLT = **RBK**+LR1*L1+LR2*L2+LR3*L3; GLT = **GBK**+LG1*L1+LG2*L2+LG3*L3; BLT = **BBK**+LB1*L1+LB2*L2+LB3*L3; R0 = R*RLT+IR0*(**RFC - **R*RLT); G0 = G*GLT+IR0*(**GFC-**G*GLT); B0 = B*BLT+IR0*(**BFC-**B*BLT);

Existing GTE command set…

✓ lighting calculations cont..

Normal ColorCol
(no depth cue)

F16,17
(x3)
NCCS
(NCCT)

L1 = L11*NXn+L12*NYn+L13*NZn;
L2 = L21*NXn+L22*NYn+L23*NZn;
L3 = L31*NXn+L32*NYn+L33*NZn;
RLT = **RBK**+LR1*L1+LR2*L2+LR3*L3;
GLT = **GBK**+LG1*L1+LG2*L2+LG3*L3;
BLT = **BBK**+LB1*L1+LB2*L2+LB3*L3;
R0 = R*RLT
G0 = G*GLT;;
B0 = B*BLT;;

Existing GTE command set...

✓ lighting calculations cont..

Material Screen Color + Depth Queuing (textured poly screen color)	**F18** CDP	RLT = **RBK**+LR1*L1^n+LR2*L2^n+LR3*L3^n; GLT = **GBK**+LG1*L1^n+LG2*L2^n+LG3*L3^n; BLT = **BBK**+LB1*L1^n+LB2*L2^n+LB3*L3^n; R0 = R*RLT+IR0*(**RFC** - R*RLT); G0 = G*GLT+IR0*(**GFC**-G*GLT); B0 = B*BLT+IR0*(**BFC**-B*BLT);

Existing GTE command set…

✓ lighting calculations cont...

Screen Color Material without Depth Cue	**F19** CC	RLT = **RBK**+LR1*L1^n+LR2*L2^n+LR3*L3^n; GLT = **GBK**+LG1*L1^n+LG2*L2^n+LG3*L3^n; BLT = **BBK**+LB1*L1^n+LB2*L2^n+LB3*L3^n; R0 = R*RLT; G0 = G*GLT; B0 = B*BLT;

Existing GTE command set…

✓ lighting subset utils

Light Source Effect	**F4** MVMVA	L1 = L11*NXn+L12*NYn+L13*NZn; L2 = L21*NXn+L22*NYn+L23*NZn; L3 = L31*NXn+L32*NYn+L33*NZn;
Light Source Color Effect (without material)	**F5** MVMVA	RLT = **RBK**+LR1*L1+LR2*L2+LR3*L3; GLT = **GBK**+LG1*L1+LG2*L2+LG3*L3; BLT = **BBK**+LB1*L1+LB2*L2+LB3*L3;
Screen Color with Depth Cue	**F6** DCPL	R0 = R*IR1+IR0*(<u>**RFC**</u> - R*IR1); G0 = G*IR2+IR0*(<u>**GFC**</u>-G*IR2); B0 = B*IR3+IR0*(<u>**BFC**</u>-B*IR3);

Existing GTE command set…

- ✓ lighting subset utils

Screen Color with Depth Cue	**F7,8 (x3)** DPCS (DPCT)	R0 = R+IR0*(**<u>RFC</u>**-R); G0 = G+IR0*(**<u>GFC</u>**-G); B0 = B+IR0*(**<u>BFC</u>**-B);
Screen Color with Interpolation	**F9** INTPL	R0 = IR1+IR0*(**<u>RFC</u>**-IR1); G0 = IR2+IR0*(**<u>GFC</u>**-IR2); B0 = IR3+IR0*(**<u>BFC</u>**-IR3);

Existing GTE command set…

✓ math utils

F10,11
(shift0&12)
SQR

```
L1^2=(L1*L1);
L2^2=(L2*L2);
L3^2=(L3*L3);
```

F20
NCLIP

```
OPZ =     SX0*SY1+SX1*SY2+SX2*SY0
          -SX0*SY2-SX1*SY0-SX2*SY1;
```

Existing GTE command set…

✓ math utils

F21
AVSZ3

$\boxed{OTZ = ZSF3*SZ0(1)+ZSF3*SZ1(2)+ ZSF3*SZ2(3);}$

Z average for 3 vertices

F22
AVSZ4

$\boxed{OTZ =ZSF4*SZx(0)+ZSF4*SZ0(1)+ZSF4*SZ1(2)+ZSF4*SZ2(3);}$

Z average for 4 vertices

Existing GTE command set…

✓ additional utils

F23,(F24)
(shift0&12)
OP

```
OPX(MAC-1, IR1) = DY1(R22)*DZ2(IR3) - DY2(IR2)*DZ1(R33);
OPY(MAC-2, IR2) = DZ1(R33)*DX2(IR1) - DZ2(IR3)*DX1(R11);
OPZ(MAC-3, IR3) = DX1(R11)*DY2(IR2) - DX2(IR1)*DY1(R22);
```

3D outer product

Existing GTE command set…

✓ additional utils

general purpose interpolation	**F25,F26** (shift0&12) GPF	IPX(MAC-1, IR1) = p(IR0)*PX0(IR1); IPY(MAC-2, IR2) = p(IR0)*PY0(IR2); IPZ(MAC-3, IR3) = p(IR0)*PZ0(IR3);
general purpose interpolation	**F27,F28** (shift0&12) GPL	IPX(MAC1, IR1) = MAC1+p(IR0)*PXn(IR1); IPY(MAC2, IR2) = MAC2+p(IR0)*PYn(IR2); IPZ(MAC3, IR3) = MAC3+p(IR0)*PZn(IR3);

Walkthrough of a basic GTE command

- ✓ RotTransPers3
  - Performs coordinate transformation of three vertices and perspective transformation.
- ✓ Please refer to GTE Command Reference ver 1.0 page 6

Walkthrough of a basic GTE command: RTPT

- Functionally you input set of 3 local coordinate vectors and obtain corresponding screen coordinates
- but wait ...

Walkthrough of a basic GTE command: RTPT...

- Make sure of the following...
  - Set your incoming local coordinate vectors.
  - Set the desired objects const rotmatrix and translation vector.
  - Set the screen offset, distance to viewpoint and depth coefficients.

Walkthrough...

- ✓ Okay now invoke RTPT
  - the results are available 23 cycles later...

Walkthrough...

✓ what exactly did RTPT do?

Calculations:

n=0,1,2{

(1,31,12)	**SSXn** = **TRX** + R11*VXn + R12*VYn + R13*VZn;	<1>
(1,31,12)	**SSYn** = **TRY** + R21*VXn + R22*VYn + R23*VZn;	<2>
(1,31,12)	**SSZn** = **TRZ** + R31*VXn + R32*VYn + R33*VZn;	<3>
(1,27,16)	**SXn** = OFX + IR1*(H / SZ n);	<4>
(1,27,16)	**SYn** = OFY + IR2*(H / SZ n);	<4>
(1,15,0)	SXn = limD1(**SXn**);	
(1,15,0)	SYn = limD2(**SYn**);	

}

(0,16,0)	SZ0(1) = limC(**SSZ0**);	
(0,16,0)	SZ1(2) = limC(**SSZ1**);	
(0,16,0)	SZ2(3) = limC(**SSZ2**);	
(0,16,0)	SZx(0) = SZ2(3);	
(1,19,24)	**P** = DQB + DQA*(H / SZ2);	
(1,3,12)	IR0 = limE(**P**);	<4>
(1,15,0)	IR1 = limA1S(**SSX2**);	
(1,15,0)	IR2 = limA2S(**SSY2**);	
(1,15,0)	IR3 = limA3S(**SSZ2**);	
(1,7,24)	**MAC0** = **P**;	
(1,31,0)	**MAC1** = **SSX2**;	
(1,31,0)	**MAC2** = **SSY2**;	
(1,31,0)	**MAC3** = **SSZ2**;vv	

Walkthrough...

- ✓ What exactly did RTPT do?

n=0,1,2{

(1,31,12) **SSXn** = **TRX** + R11*VXn + R12*VYn + R13*VZn; <1>
(1,31,12) **SSYn** = **TRY** + R21*VXn + R22*VYn + R23*VZn; <2>
(1,31,12) **SSZn** = **TRZ** + R31*VXn + R32*VYn + R33*VZn; <3>
(1,27,16) **SXn** = OFX + IR1*(H / SZ n); <4>
(1,27,16) **SYn** = OFY + IR2*(H / SZ n); <4>
(1,15,0) SXn = limD1(**SXn**);
(1,15,0) SYn = limD2(**SYn**);

}

n=0,1,2{}

Walkthrough ...

- What exactly did RTPT do?

n=0,1,2{

(1,31,12) **SSXn** = **TRX** + R11*VXn + R12*VYn + R13*VZn; <1>

(1,31,12) **SSYn** = **TRY** + R21*VXn + R22*VYn + R23*VZn; <2>

(1,31,12) **SSZn** = **TRZ** + R31*VXn + R32*VYn + R33*VZn; <3>

(1,27,16) **SXn** = OFX + IR1*(H / SZ n); <4>

(1,27,16) **SYn** = OFY + IR2*(H / SZ n); <4>

}

(1,3,12) IR0 = limE(**P**); <4>

<n>

Walkthrough...

✓ What exactly did RTPT do?

```
n=0,1,2{
        (1,31,12)   SSXn = TRX + R11*VXn + R12*VYn + R13*VZn;     <1>
        (1,31,12)   SSYn = TRY + R21*VXn + R22*VYn + R23*VZn;     <2>
        (1,31,12)   SSZn = TRZ + R31*VXn + R32*VYn + R33*VZn;     <3>
        (1,27,16)   SXn = OFX + IR1*(H / SZ n);        <4>
        (1,27,16)   SYn = OFY + IR2*(H / SZ n);        <4>
        (1,15,0)    SXn = limD1(SXn);
        (1,15,0)    SYn = limD2(SYn);
}
(0,16,0)    SZ0(1) = limC(SSZ0);
(0,16,0)    SZ1(2) = limC(SSZ1);
(0,16,0)    SZ2(3) = limC(SSZ2);
(1,19,24)   P = DQB + DQA*(H / SZ2);
(1,3,12)    IR0 = limE(P);          <4>
(1,15,0)    IR1 = limA1S(SSX2);
(1,15,0)    IR2 = limA2S(SSY2);
(1,15,0)    IR3 = limA3S(SSZ2);
(1,7,24)    MAC0 = P;
(1,31,0)    MAC1 = SSX2;
(1,31,0)    MAC2 = SSY2;
(1,31,0)    MAC3 = SSZ2;
```

VAL

Walkthrough...

✓ What exactly did RTPT do?

n=0,1,2{
(1,31,12) **<u>SSXn</u>** = **TRX** + R11*VXn + R12*VYn + R13*VZn; <1>
(1,31,12) **<u>SSYn</u>** = **TRY** + R21*VXn + R22*VYn + R23*VZn; <2>
(1,31,12) **<u>SSZn</u>** = **TRZ** + R31*VXn + R32*VYn + R33*VZn; <3>
}

OBJ

Walkthrough...

✓ What exactly did RTPT do?

```
n=0,1,2{
        (1,15,0)   SXn = limD1(SXn);
        (1,15,0)   SYn = limD2(SYn);
}
(0,16,0)   SZ0(1) = limC(SSZ0);
(0,16,0)   SZ0(2) = limC(SSZ1);
(0,16,0)   SZ0(3) = limC(SSZ2);
(1,3,12)   IR0 = limE(P);      <4>
(1,15,0)   IR1 = limA1S(SSX2);
(1,15,0)   IR2 = limA2S(SSY2);
(1,15,0)   IR3 = limA3S(SSZ2);
```

limX()

Walkthrough...

✓ What exactly did RTPT do?

n=0,1,2{

(1,15,0) SXn = limD1(**SXn**);

(1,15,0) SYn = limD2(**SYn**);

}

(0,16,0) SZ0(1) = limC(**SSZ0**);

(0,16,0) SZ0(2) = limC(**SSZ1**);

(0,16,0) SZ0(3) = limC(**SSZ2**);

(1,3,12) IR0 = limE(**P**);

(1,15,0) IR1 = limA1S(**SSX2**);

(1,15,0) IR2 = limA2S(**SSY2**);

(1,15,0) IR3 = limA3S(**SSZ2**);

(1,7,24) **MAC0** = **P**;

(1,31,0) **MAC1** = **SSX2**;

(1,31,0) **MAC2** = **SSY2**;

(1,31,0) **MAC3** = **SSZ2**;

results

Walkthrough...

✓ corresponding assembler source...

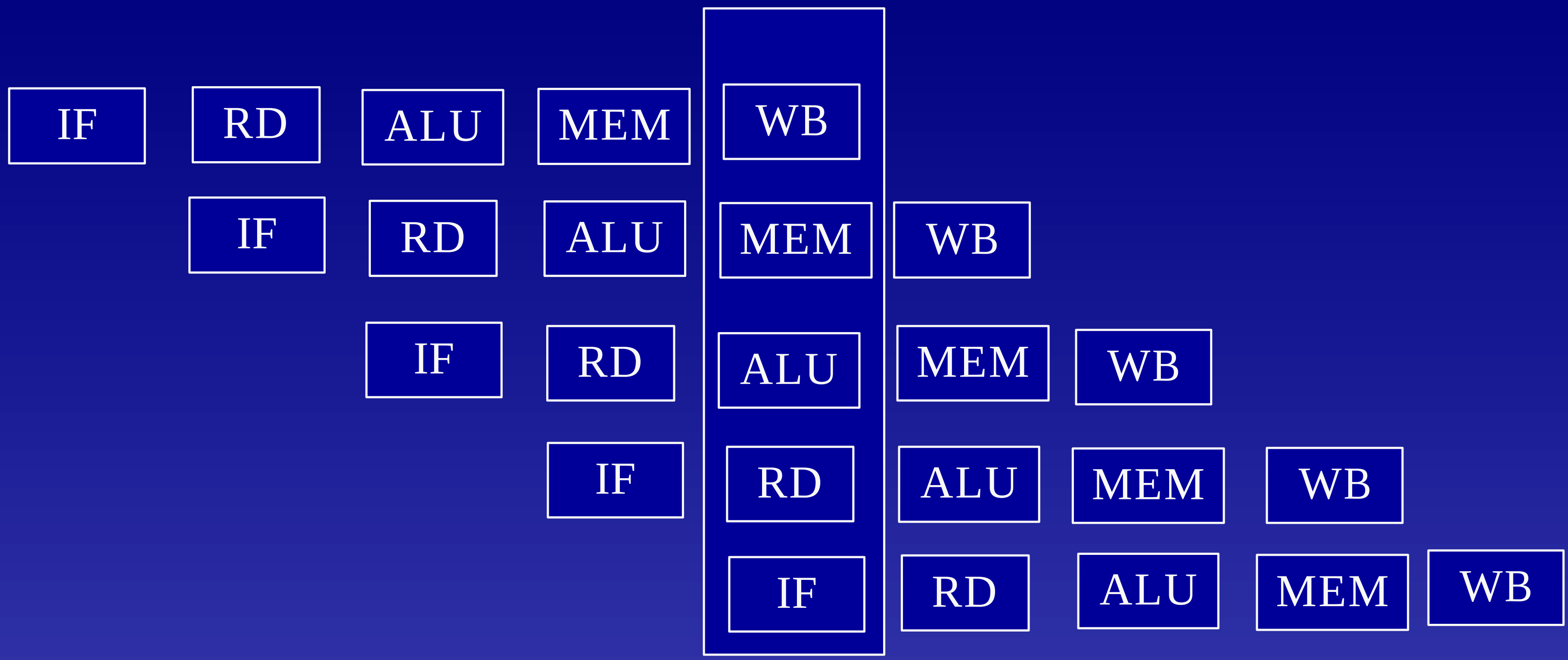
```
                .globl      RotTransPers3
                .text
                .set        noat
RotTransPers3:
                .set        noreorder
                lwc2        C2_VXY0,(a0)
                lwc2        C2_VZ0,4(a0)
                lwc2        C2_VXY1,(a1)
                lwc2        C2_VZ1,4(a1)
                lwc2        C2_VXY2,(a2)
                lwc2        C2_VZ2,4(a2)
                nop
                RTPT
                lw          t0,16(sp)
                lw          t1,20(sp)
                lw          t2,24(sp)
                lw          t3,28(sp)
                swc2        C2_SXY0,(a3)
                swc2        C2_SXY1,(t0)
                swc2        C2_SXY2,(t1)
                swc2        C2_IR0,(t2)
                cfc2        v1,C2_FLAG0
                mfc2        v0,C2_SZ2
                sw          v1,(t3)
                j           ra
                sra         v0,v0,2
                .set        reorder
```

input local vector

output screen coordinates

GTE Machine Language Programming

CPU Pipeline



GTE machine language programming...

- Handling of delay slots
  - Insertion of dangerous commands into delay slot
  - Careless deletion of nop

Be careful with programs that appear to work correctly on the surface

GTE machine language programming tips

```
cfc2 v0,C2_FLAG0
nop
and v0, v0,v1
        :
        :
```

CORRECT

```
cfc2 v0,C2_FLAG0
and v0, v0,v1
        :
        :
```

INCORRECT

Some additional caveats

- ✓ GTE instructions cannot be used in exception handler.
- ✓ GTE instructions (mfc2, mtc2, cfc2, ctc2, lwc2, swc2) cannot be used in branch delay slot.
- ✓ load instructions(lwc2, mtc2, ctc2) cannot be used between cop2(GTE) and save instructions(swc2, mfc2, cfc2)

Some additional caveats…

- ✓ If the destination register of load instruction is not used in the cop2(GTE), it is possible to use load instructions between cop2 and save instruction

Some additional caveats…

- more examples...

```
RTPT                    /* GTE instr*/
  :          /* cpu inst */
mtc2 v0,C2_RGB/* OK */
   :         /* cpu instr */
cfc2 v0,C2_FLAG0 /* save */
```

CORRECT

```
RTPT                    /*GTE instr */
    :        /*cpu instr */
mtc2 v0,C2_VXY0 /* BAD */
    :        /* cpu instr */
cfc2 v0,C2_FLAG0 /* save */
```

INCORRECT

Some additional caveats…

✓ more examples...

```
RTPS                /* GTE instr*/
  :         /* cpu inst */
mtc2 v0,C2_VXY1/* OK */
   :        /* cpu instr */
NCLIP    /* GTE instr */
```

CORRECT

```
RTPS                /*GTE instr */
    :       /*cpu instr */
mtc2 v0,C2_VXY0 /* BAD */
    :       /* cpu instr */
NCLIP               /*GTE instr */
```

INCORRECT

More examples...

```
RTPS
## interlock
cfc2 v0,C2_FLAG
```

CPU interlock for 15 cycles

```
RTPS
add v1,v2,v3
sub v1,v2,v3
## interlock
cfc2 v0,C2_FLAG
```

CPU interlock for 13 cycles

thus ...

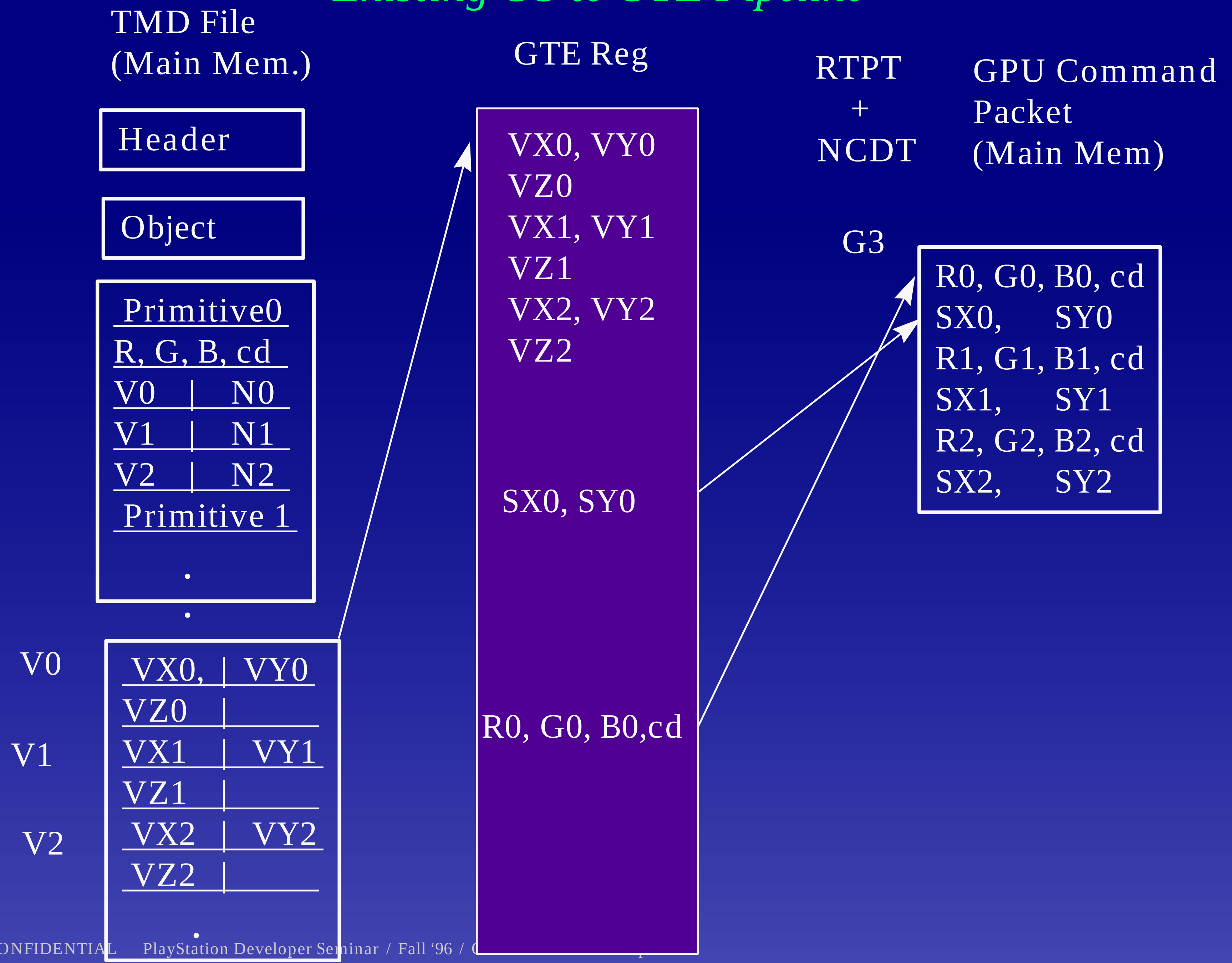
- ✓ develop your programs in the following order

libgte → DMPSX(C) → DMPSX (assembler)

Conclusion

- ✓ significance of all this information

Existing GS to GTE Pipeline



Conclusion...

- ✓ one immediate speedup would be to describe quadrilaterals as two connected triangles

Quadrilateral as two connected triangles

TMD primitive (G3x2)

G3

Primitive0	
R, G, B, cd	
V0	N0
V1	N1
V2	N2

G3

Primitive 1	
R, G, B, cd	
V1	N1
V2	N2
V3	N3

Primitive: 4x2 = 8 words
Vector: 6x2x2 = 24 words
32 words

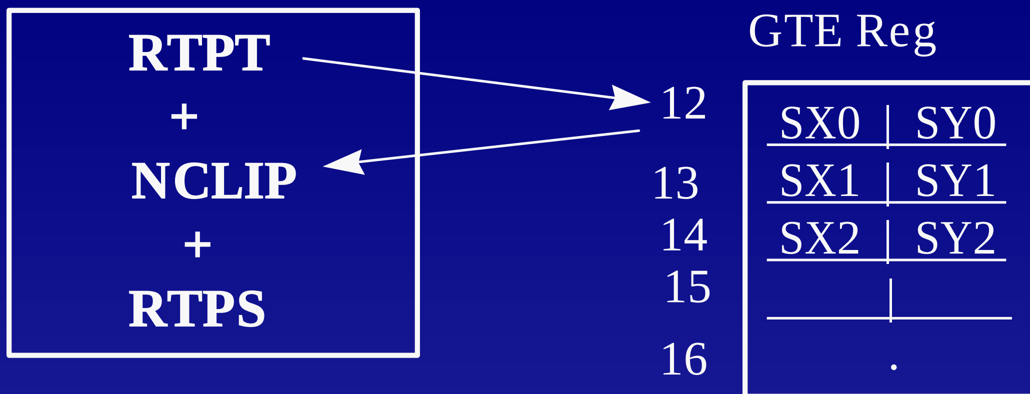
TMD primitive (G4)

G4

Primitive0	
R, G, B, cd	
V0	N0
V1	N1
V2	N2
V3	N3

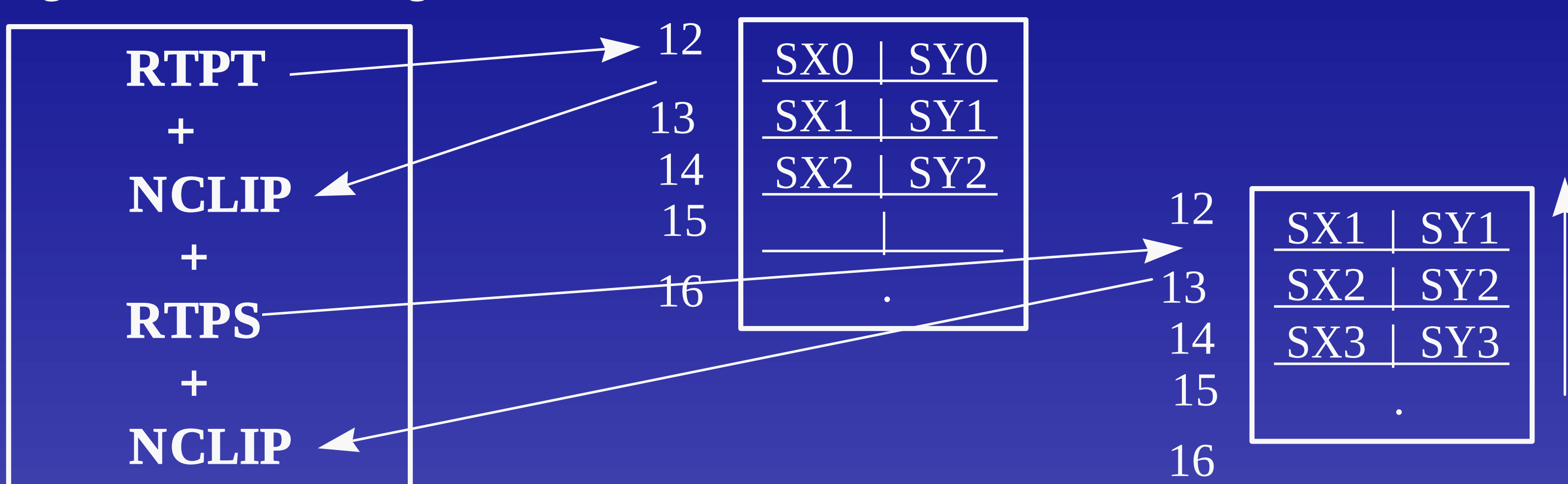
Primitive: 5 = 5 words
Vector: 8x2 = 16 words
21 words

Quadrilateral...

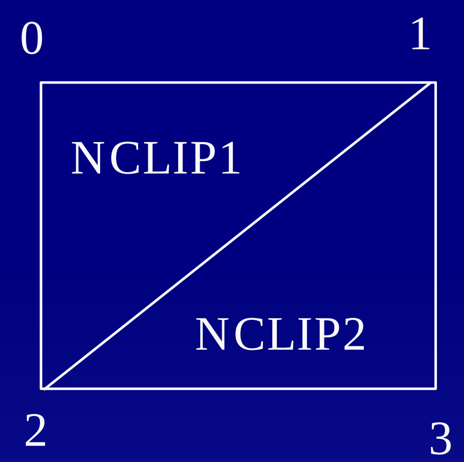


unable to do NCIP
on last vertex

Using connected triangles instead



Quadrilateral...



G4

Primitive0	
R, G, B, cd	
V0	N0
V1	N1
V2	N2
V3	N3

G4 + NCLIP x2

GPU Packet
G4

R0, G0, B0, cd
SX0, SY0
R1, G1, B1, cd
SX1, SY1
R2, G2, B2, cd
SX2, SY2
R3, G3, B3, cd
SX3, SY3

R0, G0, B0, cd
SX0, SY0
R1, G1, B1, cd
SX1, SY1
R2, G2, B2, cd
SX2, SY2

GPU Packet
G3

GPU Packet
G3

R1, G1, B1, cd
SX1, SY1
R2, G2, B2, cd
SX2, SY2
R3, G3, B3, cd
SX3, SY3

End of Part 1

PART 2: DMPSX 3.01

DMPSX 3.01 Overview

- ✓ What is it?
  - A tool for three level optimization of GTE commands

DMPSX 3.01 components

- ✓ GTEMAC.H
  - A series of replacement macros for most GTE functions

```
#define gte_RotTransPers(r1,r2,r3,r4,r5)            \
                { gte_ldv0(r1);                     \
                  gte_rtps();                       \
                  gte_stsxy(r2);                    \
                  gte_stdp(r3);                     \
                  gte_stflg(r4);                    \
                  gte_stszotz(r5);
                }
```

DMPSX 3.01 components...

- INLINE_C.H
  - Assembly macros for subcomponents of larger macros in GTEMAC.H

```
#define gte_rtps() {              \
"nop;"                            \
"nop;"                            \
".word 0x0000007f" )
}
```

DMPSX 3.01 components...

- INLINE_A.H
  - Macro definitions for assembler programs

```
nRTPS
  macro
  nop
  nop
  dw          $0000007f
  endm
```

DMPSX 3.01 components...

- GTEREG.H
  - GTE registers macros for assembler programs
- INLINE_O.H
  - Dummy macros from older version of DMPSX

DMPSX 3.01
Optimization Level 1

- ✓ Designed to help programs run within the I-cache
- ✓ Programs which currently run on within the I-cache may experience slow-down

DMPSX 3.01
Optimization Level 1 (cont.)

- Replace functions found in gtemac.h
  - Prefix function with "gte_"

    RotTransPers() → gte_RotTransPers()

  - Add return value to end of argument list

    otz=RotTransPers() → gte_RotTransPers(...,&otz)

  - If GTE constants destroyed, save and load these constants

    OuterProduct12() → gte_ReadRotMatrix(&m)
    gte_OuterProduct12()
    gte_SetRotMatrix(&m)

DMPSX 3.01
Optimization Level 1 (cont.)

- ✓ Write inline functions directly in the program

RotTransPers4() → gte_RptTransPers3()
gte_RotTransPers()

DMPSX 3.01
Optimization Level 2

Use the sub-macros in gtemac.h to delete unneeded GTE commands

```
{
    gte_ldv0(v0);
    gte_rtps();
    gte_stsxy(sxy);
    gte_stdp(p);
    gte_stflg(flag);
    gte_stszotz(otz);
}
```

{
gte_ldv0(v0);
gte_rtps();
gte_stsxy(sxy);
~~gte_stdp(p);~~
~~gte_stflg(flag)~~;
gte_stszotz(otz);
}

DMPSX 3.01 Optimization Level 3

✓ Insert R3000 commands

- Three types of GTE commands:

Type 1: Load GTE register	Fast
Type 2: Execute GTE instruction	Slow
Type 3: Read GTE register	Fast

Example:

```
{
    gte_ldv0(v0);        Type 1
    gte_rtps();          Type 2
    gte_stsxy(sxy);      Type 3
    gte_stszotz(otz);    Type 3
}
```

DMPSX 3.01
Optimization Level 3 (cont.)

✓Insert R3000 commands

```
{
    gte_ldv0(v0);
    gte_rtps();
    /* Type 2 = wait for GTE */
    gte_stsxy(sxy);
    gte_stszotz(otz);
}
```

⇒

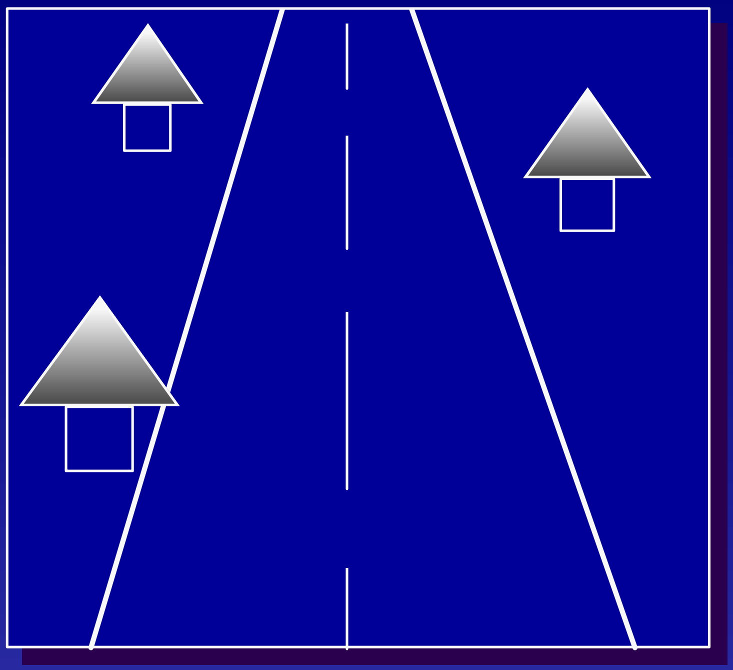
```
{
    gte_ldv0(v0);
    gte_rtps();
    R3000 Process
    gte_stsxy(sxy);
    gte_stszotz(otz);
}
```

End of Part 2

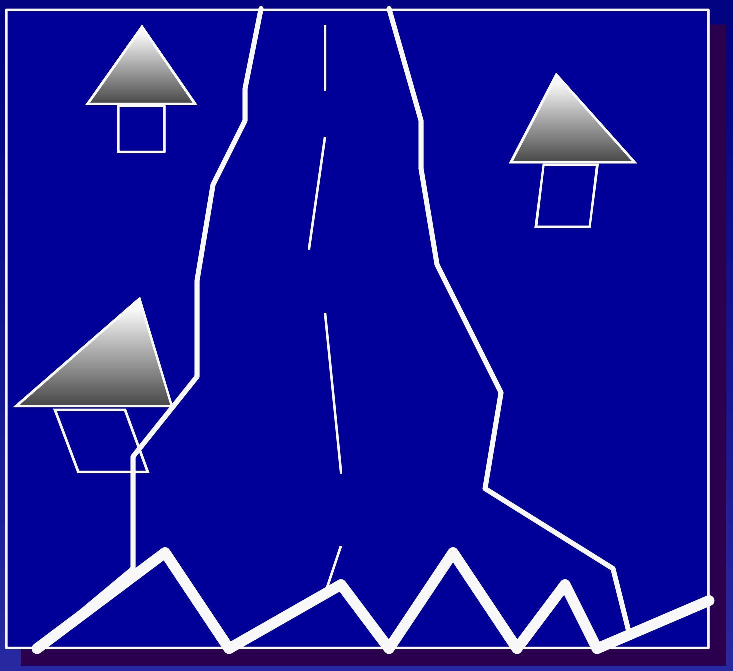
Part 3:
Revisiting Some Old Favorites

Methods for speeding up polygon division

Problems involved in displaying ground



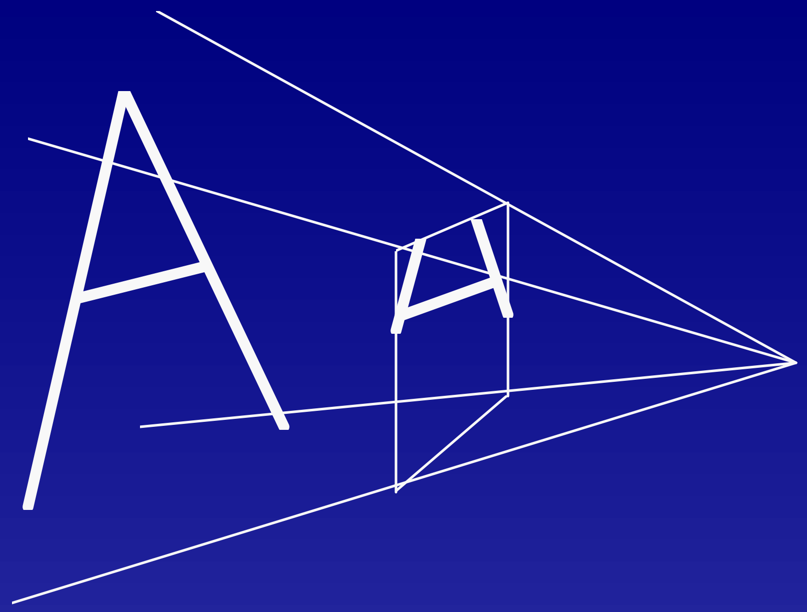
Intended result



1. Warping of texture

2. Near clipping problems

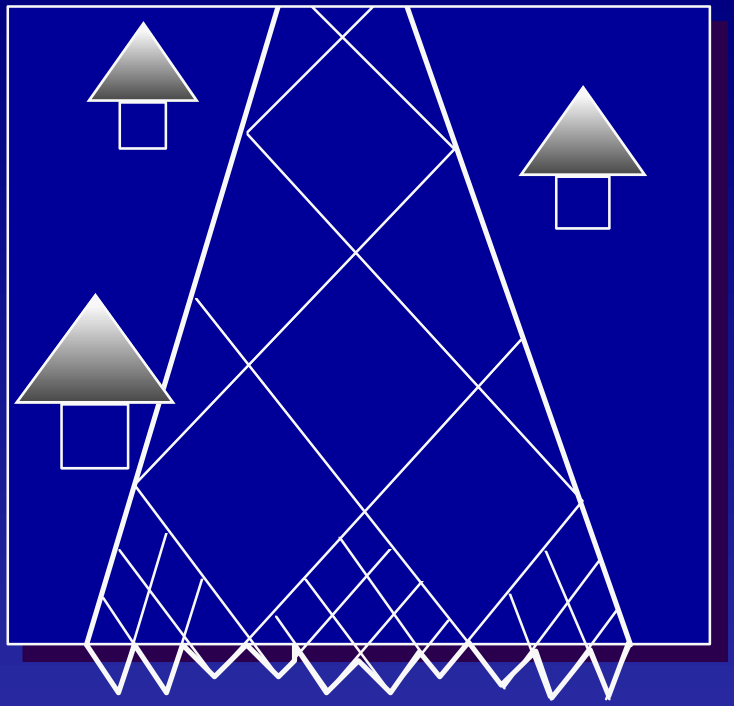
Solution using clipping



O allows more polygons to be used

X texture jumping
X texture warping
X calculations become more complex

Solution using division



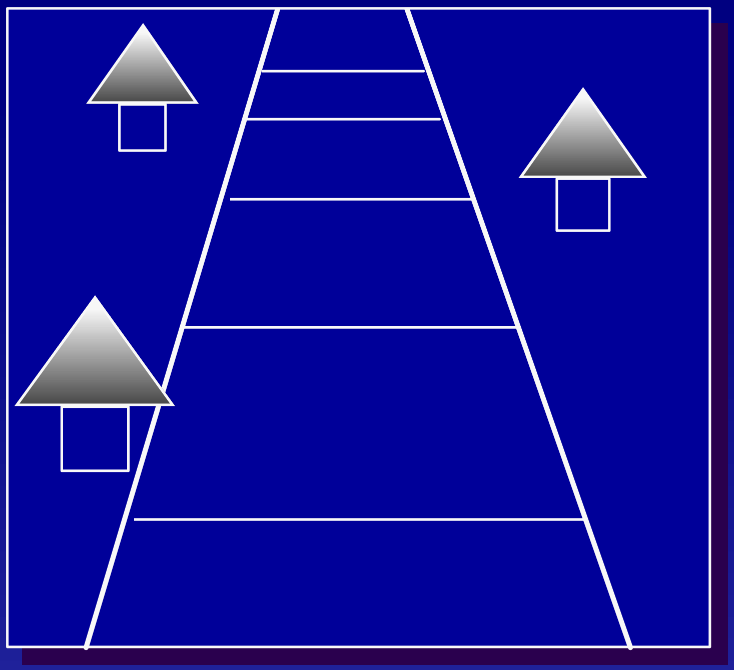
O less texture jumping

O texture warping is eliminated

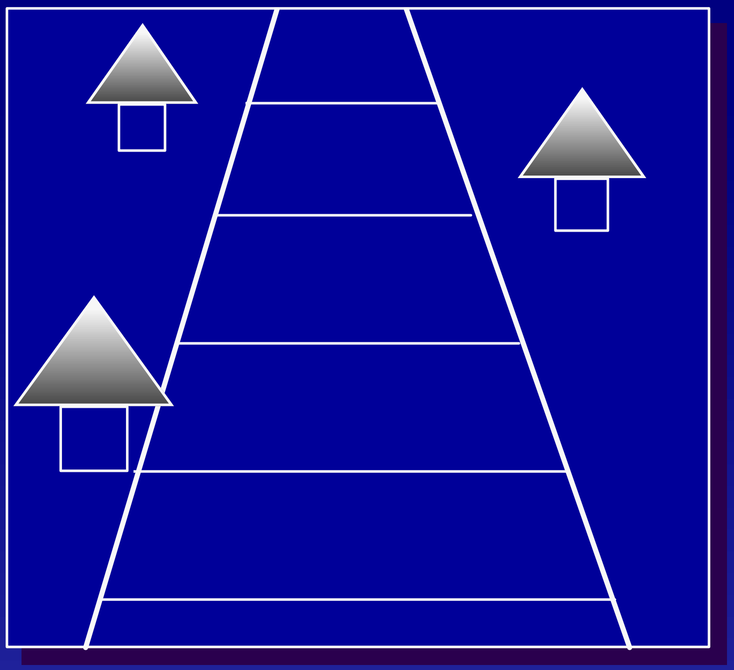
X the polygon count is increased

Using the division method is better!

Divide in 2 dimensions or 3 dimensions?



3 dimensions

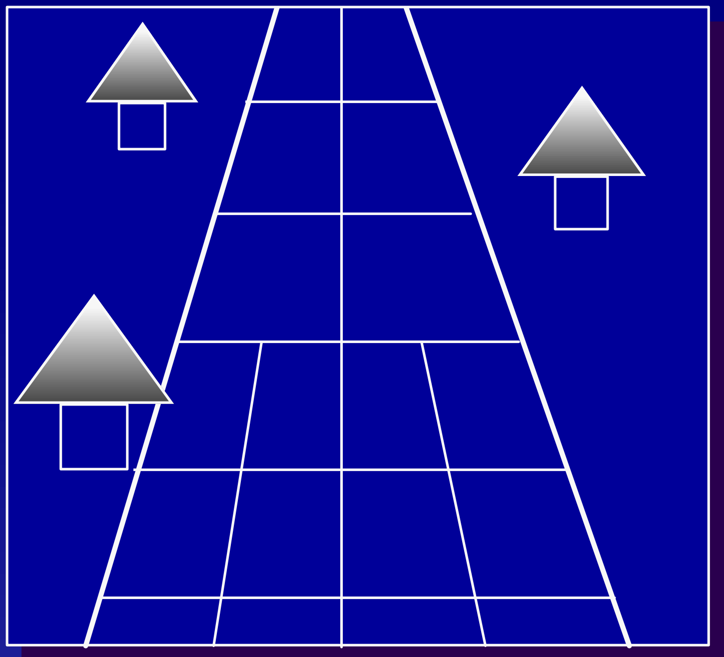


2 dimensions

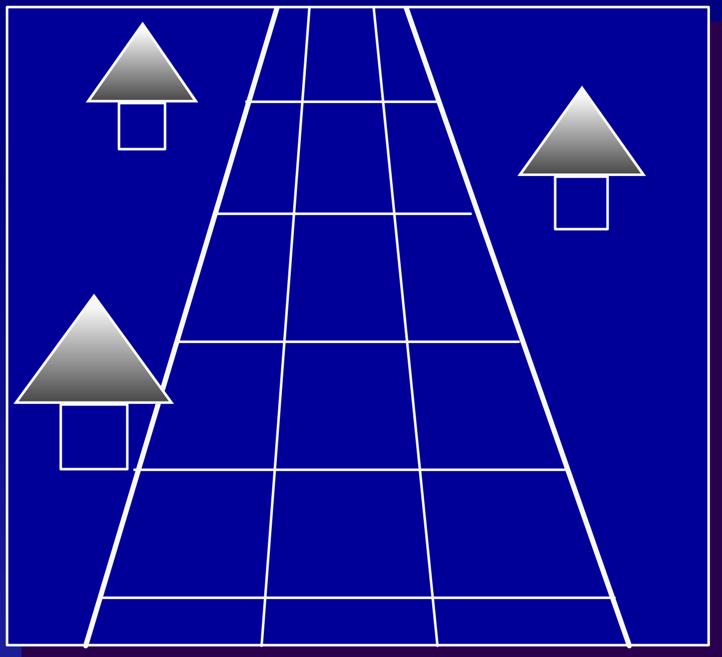
Divide in three dimensions

- 3 dimensions provides more accuracy
- Because GTE calculations are performed at high speeds, there is no overhead with 3-dimensional division

Active division or fixed division?



Active



Fixed

Use active method

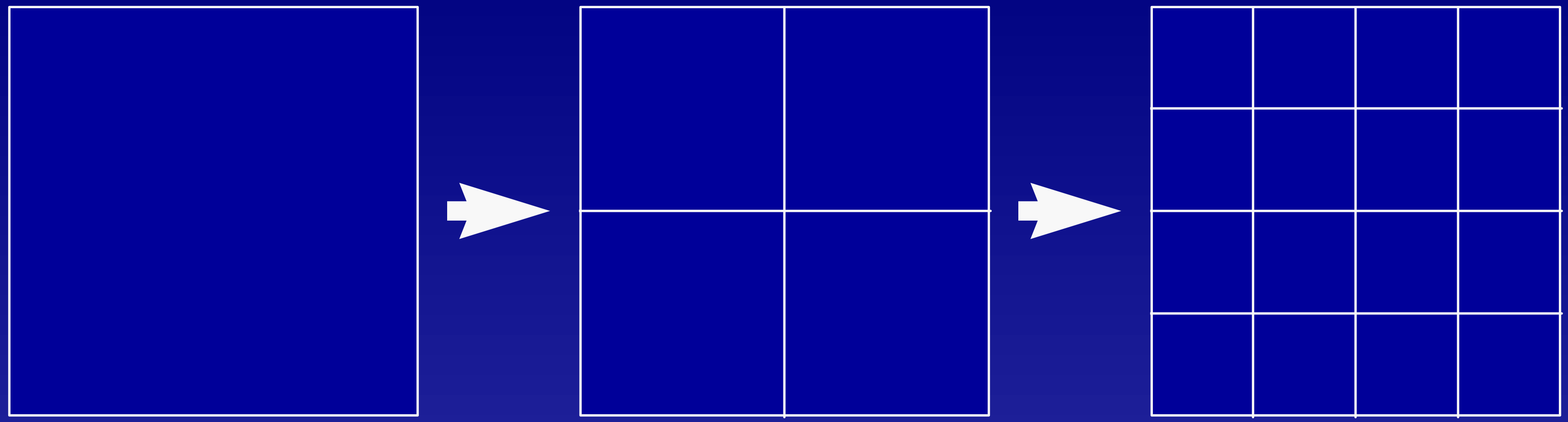
Advantages		**Disadvantages**
1. Polygon count is decreased	⟷	1. Gaps are generated
2. Improves speed		2. Textures become non- continuous

Actual programming

Principle

Display ground using active, 3-dimensional division

Recursive call



2^n **division**

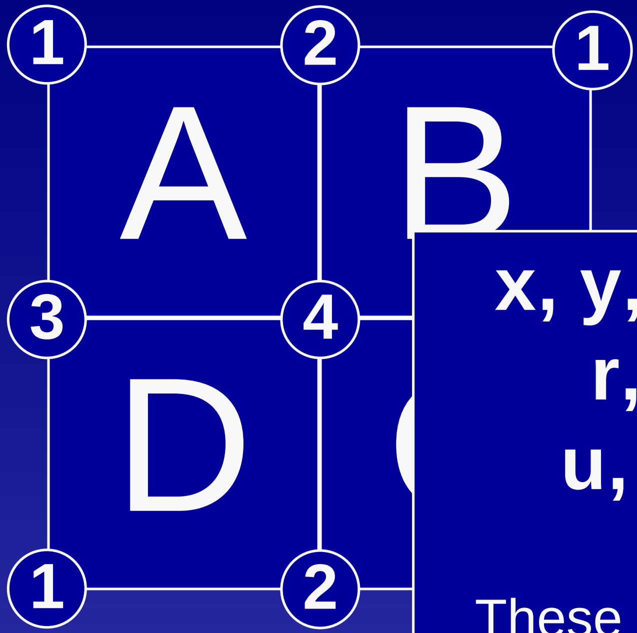
Conditions for stopping

<Polygon vertex distance>

Reasons

- GPU rendering limit 1024x512
- Polygon warping is most noticeable with larger polygons
- Used together with Area Clipping

3-Dimensional 2n division



ordered as follows: A->B->C->D

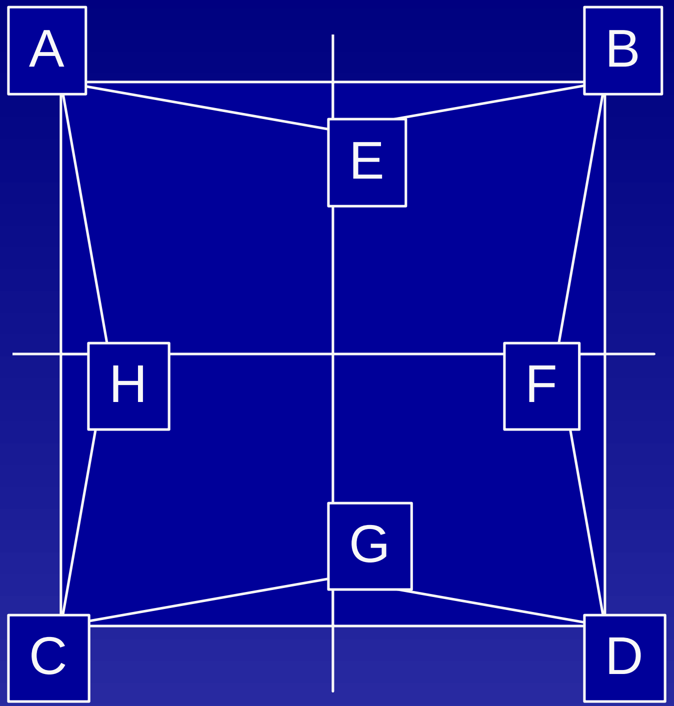
x, y, z coordinates

r, g, b color

u, v texture

These are all divided by two

Fixing gaps



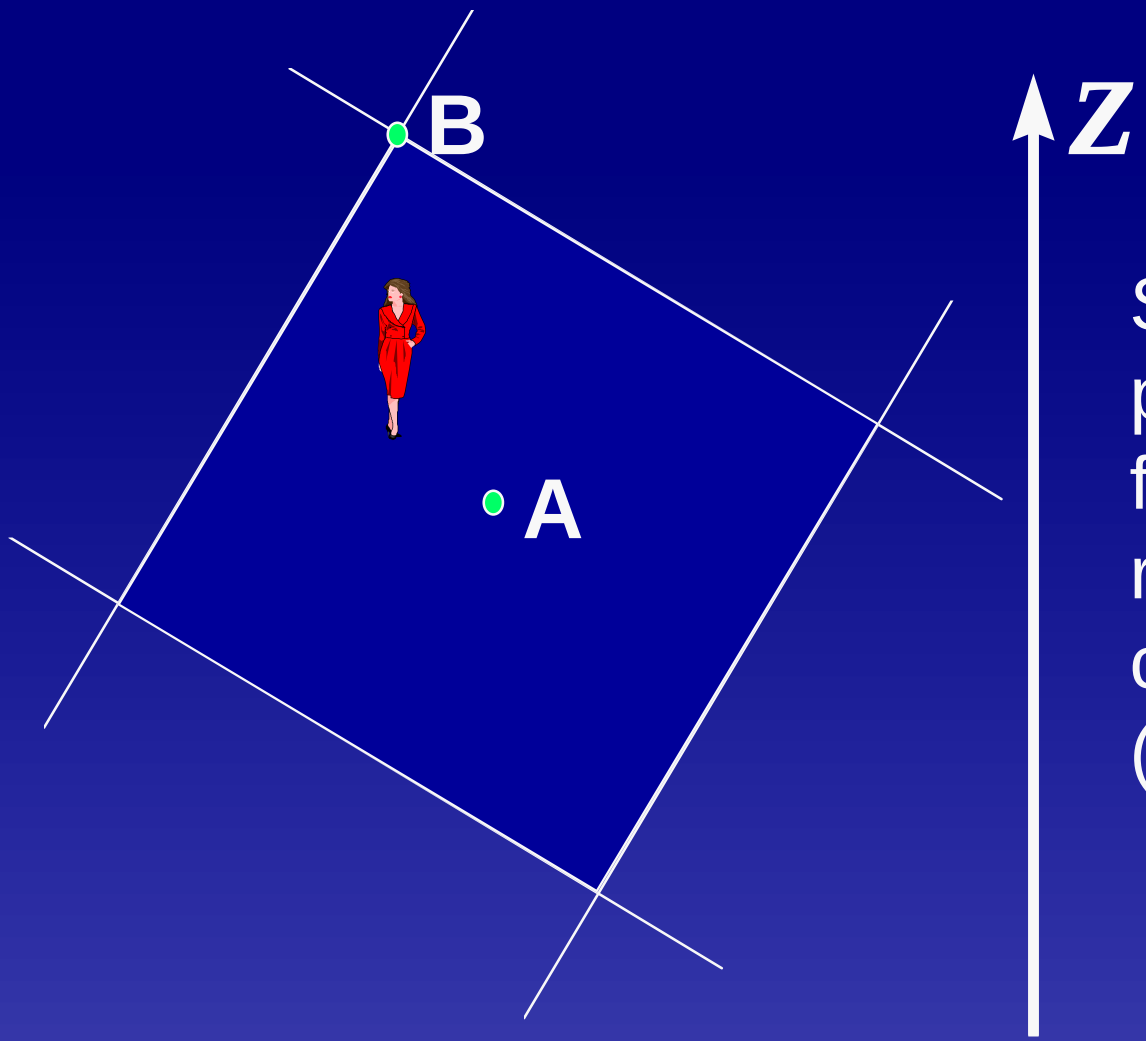
Reason

Due to the margin of error, the center point does not necessary lie on the axis

Solution

Draw a triangle for the gap as well
However, Back Clip is necessary

Solving the Z-sort problem

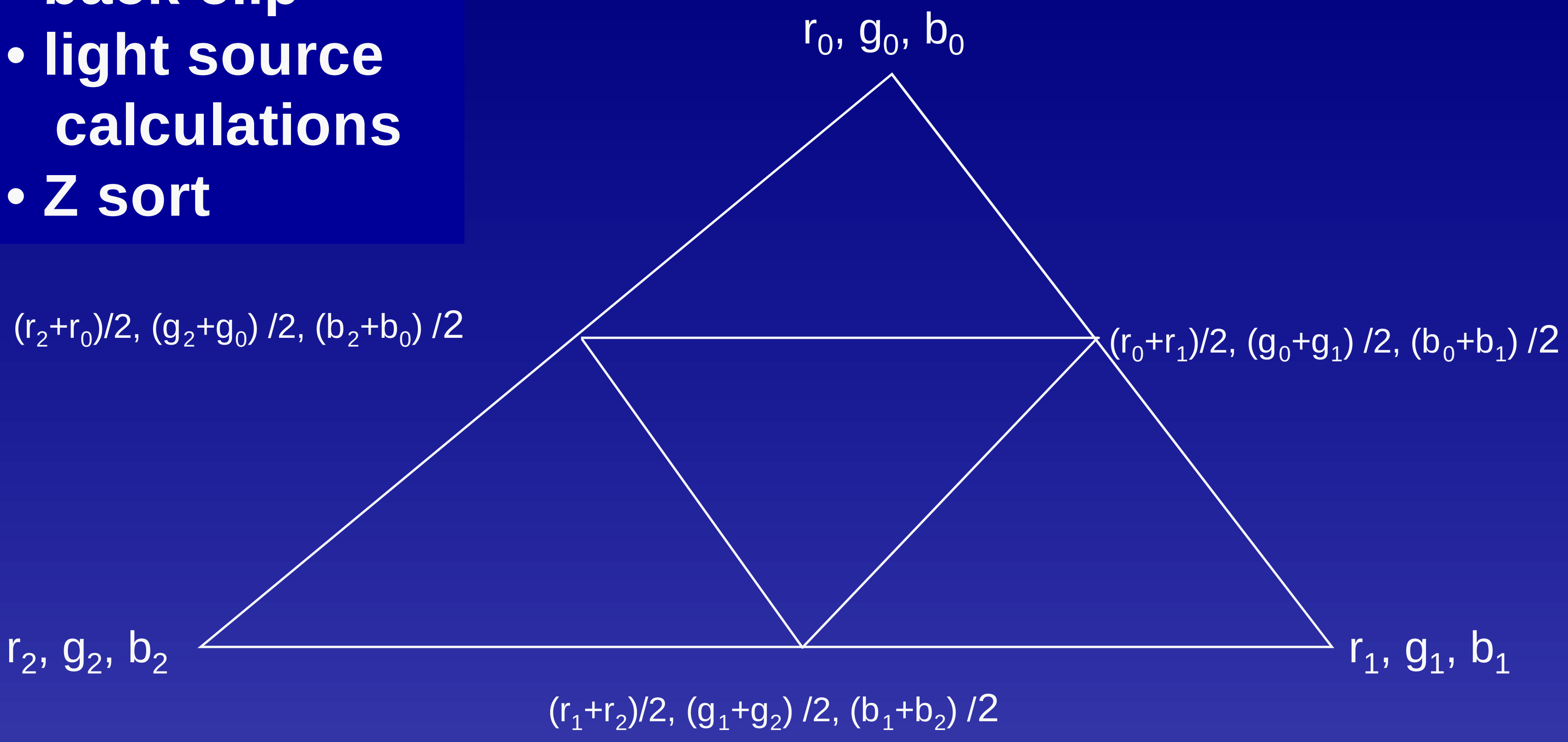


Set the Z-sort point to the furthest point (B) rather than the center of gravity (A)

Split processing for before and after division

Processing that is performed just once before division

- **back clip**
- **light source calculations**
- **Z sort**



READ modeling data

(x_0, y_0, z_0)

(x_1, y_1, z_1)

(x_2, y_2, z_2)

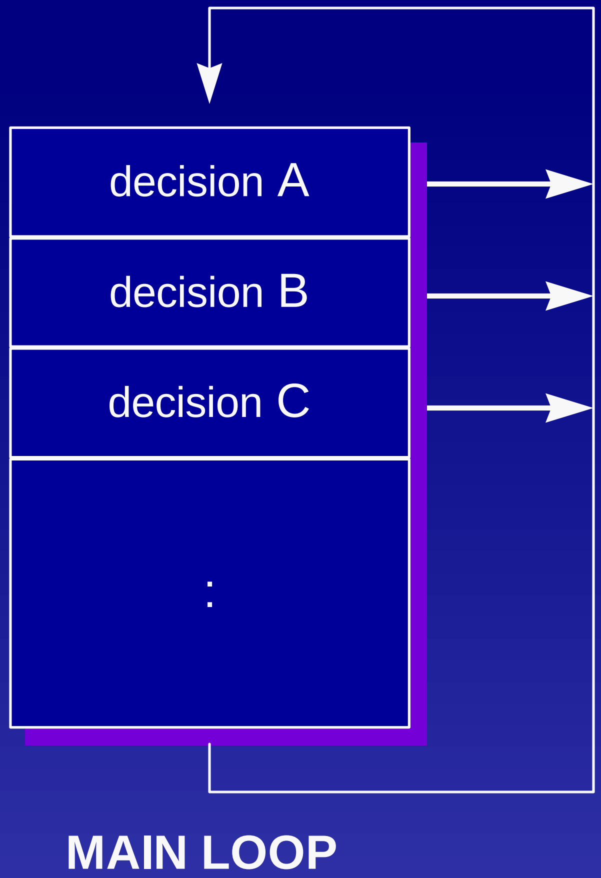
(x_3, y_3, z_3)

Reading 4 vertices, 12 words takes about 70 cycles

If the data can be expressed as w,h,d, about 7 words and 20 cyclescan be saved

Modeling data formats should take into consideration the fact that memory reads are very slow

Polygons that will not be displayed should be rejected early on



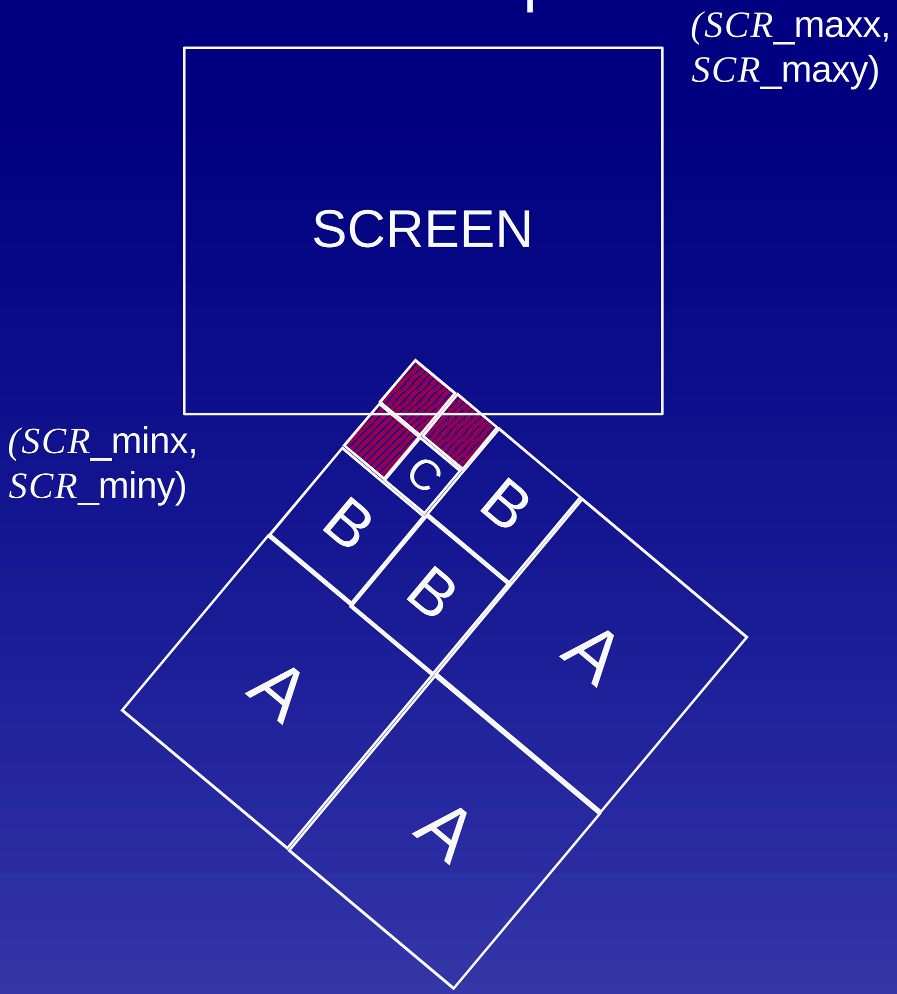
the rejection amount is

A > B > C

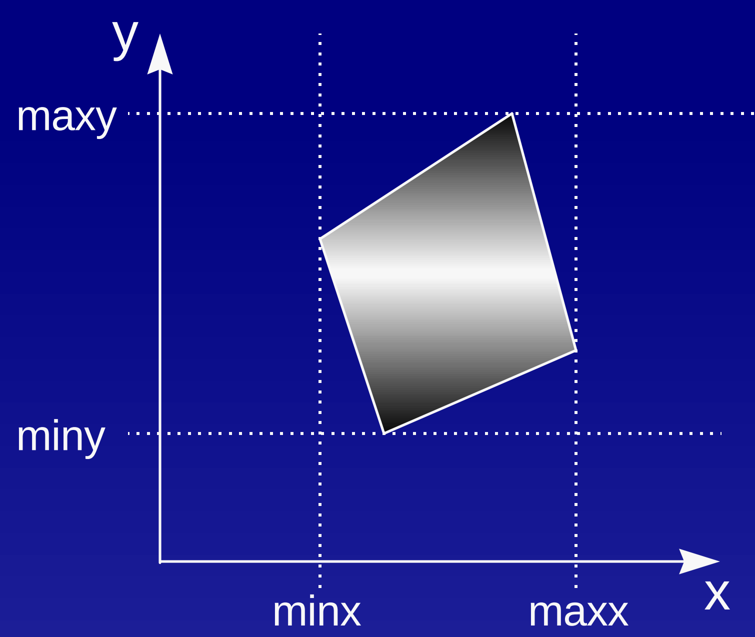
A is the GTE flag clip

Clipping (1)

HW clip



4-vertex **min-max**

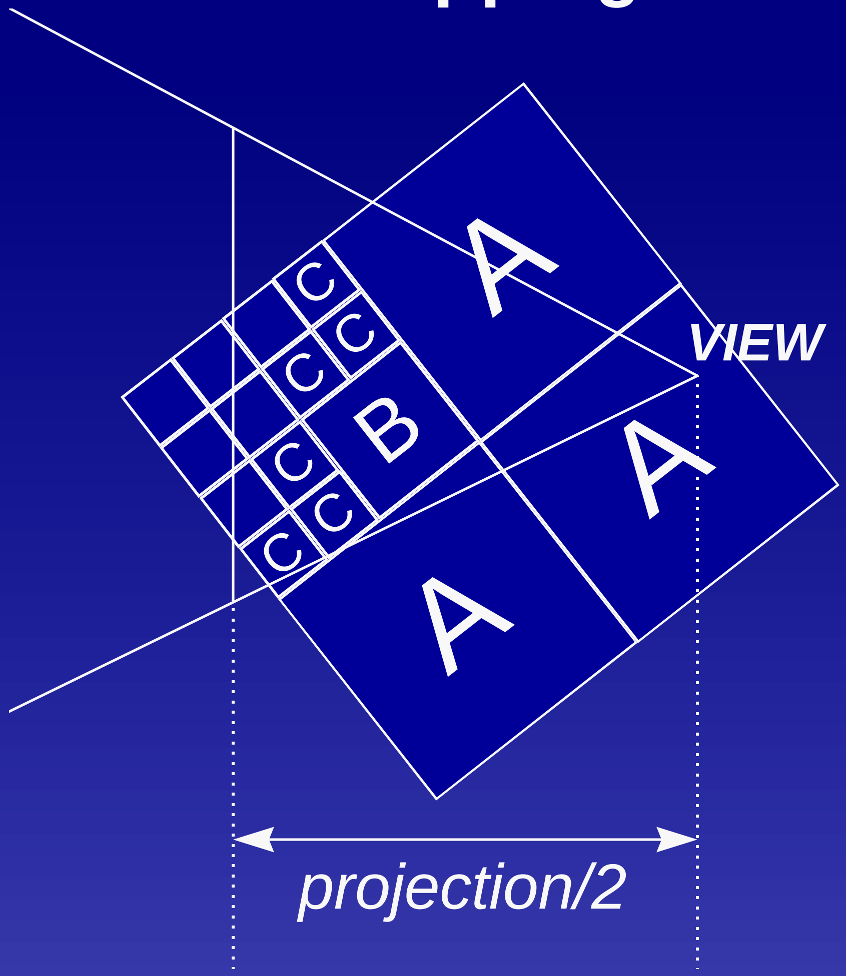


Clip conditions

maxx	>	*SCR*_minx
maxy	>	*SCR*_miny
minx	>	*SCR*_maxx
miny	>	*SCR*_maxy

Clipping (2)

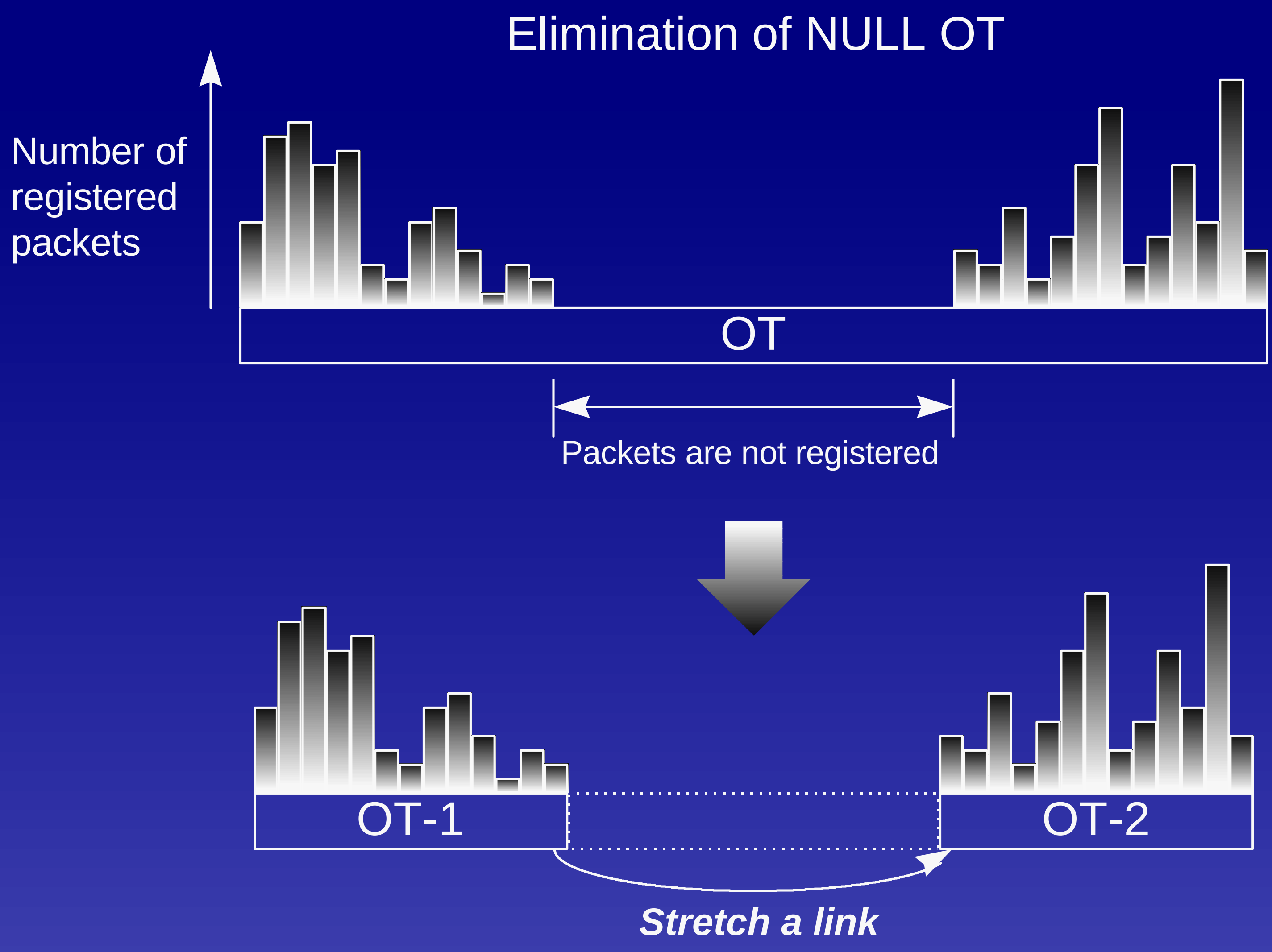
NEAR **Z clipping**



Clip conditions

SZ0 < *projection/2*
&
SZ1 < *projection/2*
&
SZ2 < *projection/2*
&
SZ3 < *projection/2*

Eliminating useless OT



Conclusion

Rendering ground in 3-dimensions

1. Active 3-dimension divisions
2. Recursive call
3. On cache

End