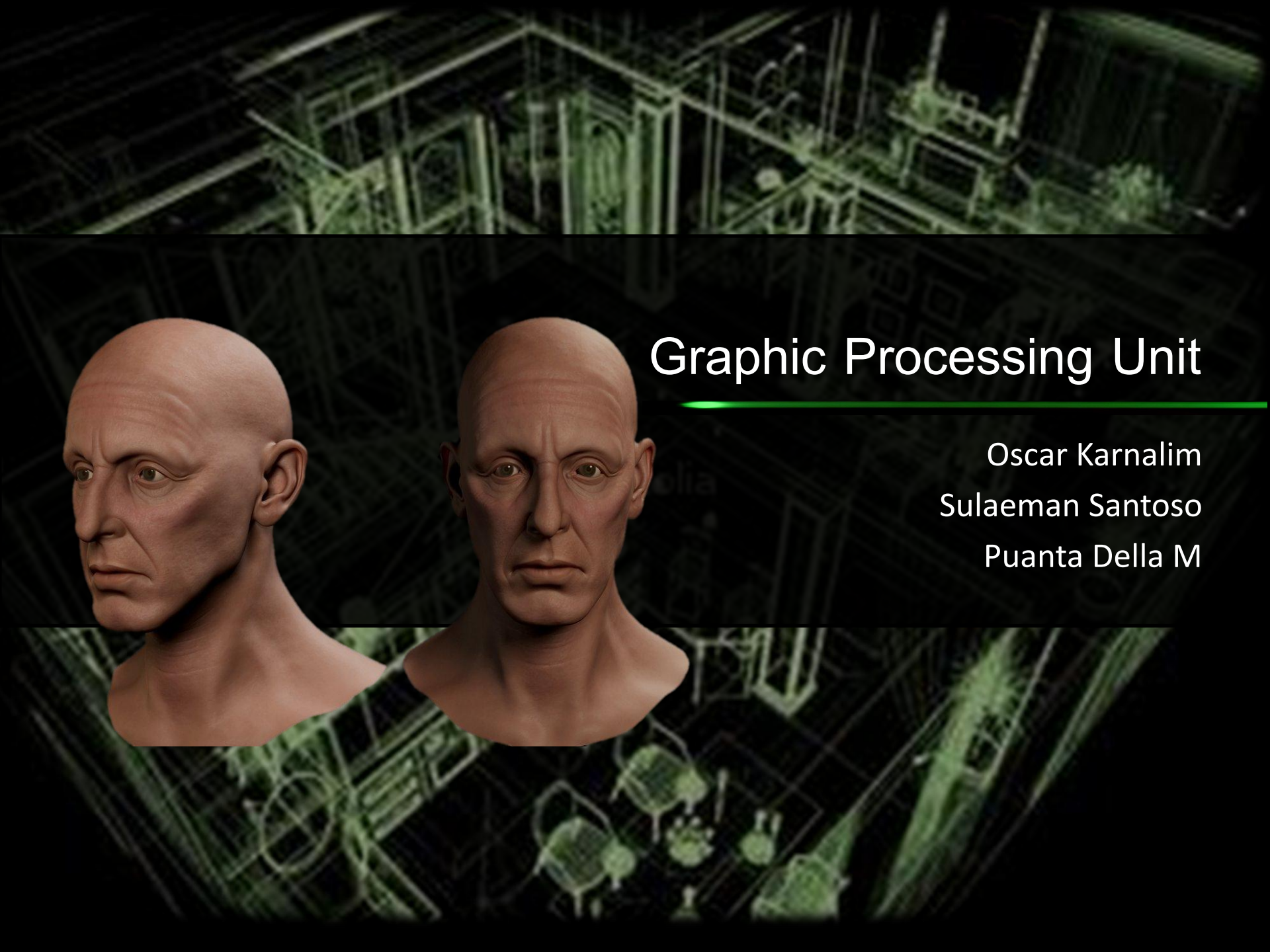


Graphic Processing Unit

Oscar Karnalim
Sulaeman Santoso
Puanta Della M

Why GPU ?

- Computer Graphics demands are skyrocketing
  - Games, Movies, Simulation, etc
- Better graphics = more computation
- CPU is not sufficient to deal with graphic demands

A little example

- A single 3D human model requires 10.000 – 12.0000 vertices
- To animate or renders a single model requires a multiplication of that 10k-12k vertices
- A modern MMORPG can handle battles between 200+ player

how much computation would

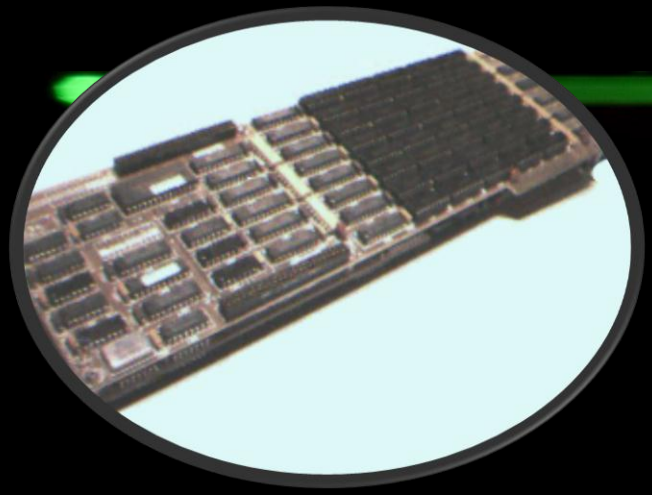
a scene like this required ?

GPU to the rescue

- Graphics Processing Unit
- Unit specially designed to handle graphic computation
- Special architecture to achieve efficient computation
- Efficient at floating point operation and matrix computation

GPU

A history of GPU



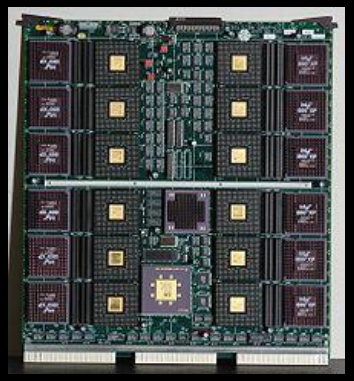
- 1980 : IBM’s PGA

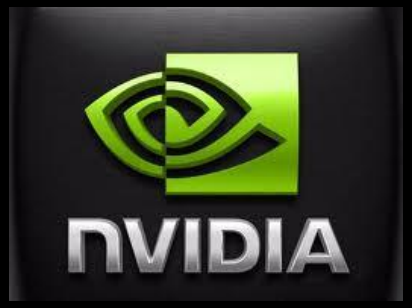
- 1989 : OPEN GL release
  pipeline graphics beginning

A history of GPU (2)

- 1993 : Sillicon Graphic's Reality Engine

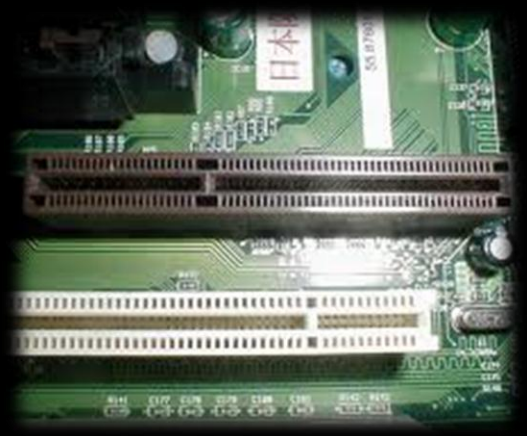
  3DFX, Nvidia, ATI, Matrox

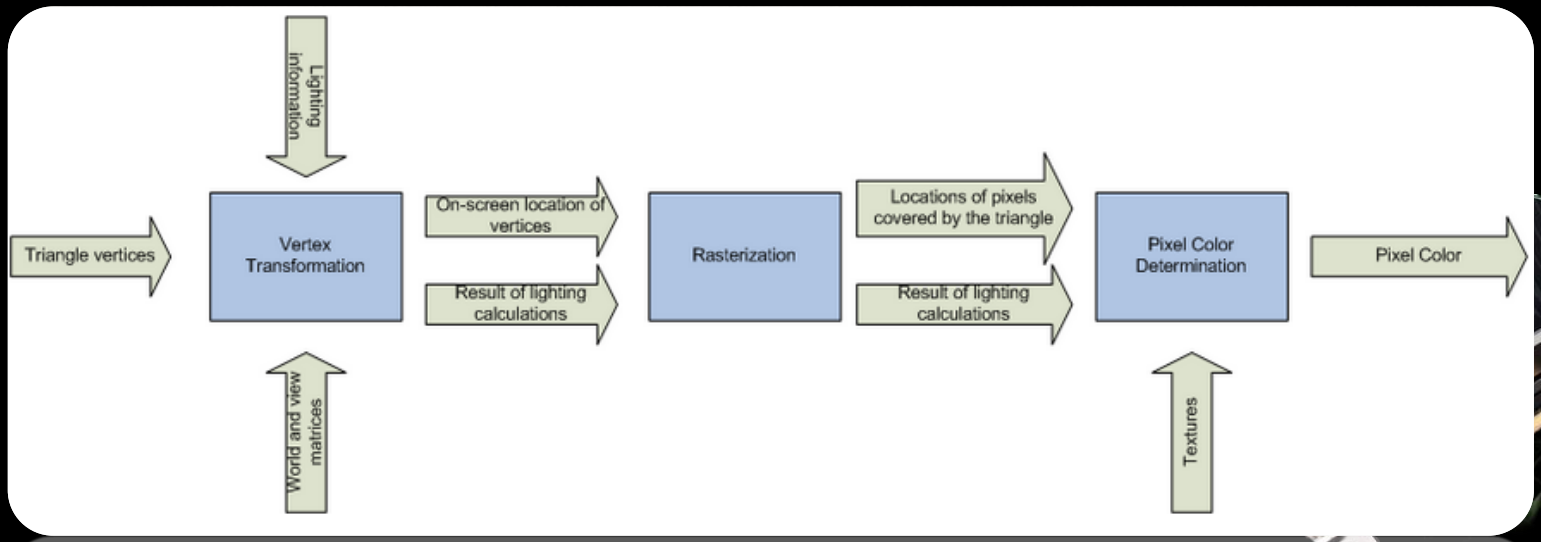




A history of GPU (3)

- 1999 : full hardware pipeline
  AGP port
  the term GPU is invented
  fixed function pipeline





A history of GPU (4)

- 2001 : Programmable pipeline
  shader program

A history of GPU (5)

- 2002 : Fully programmable pipeline
  pixel shader, vertex shader
  DirectX 9

A history of GPU (6)

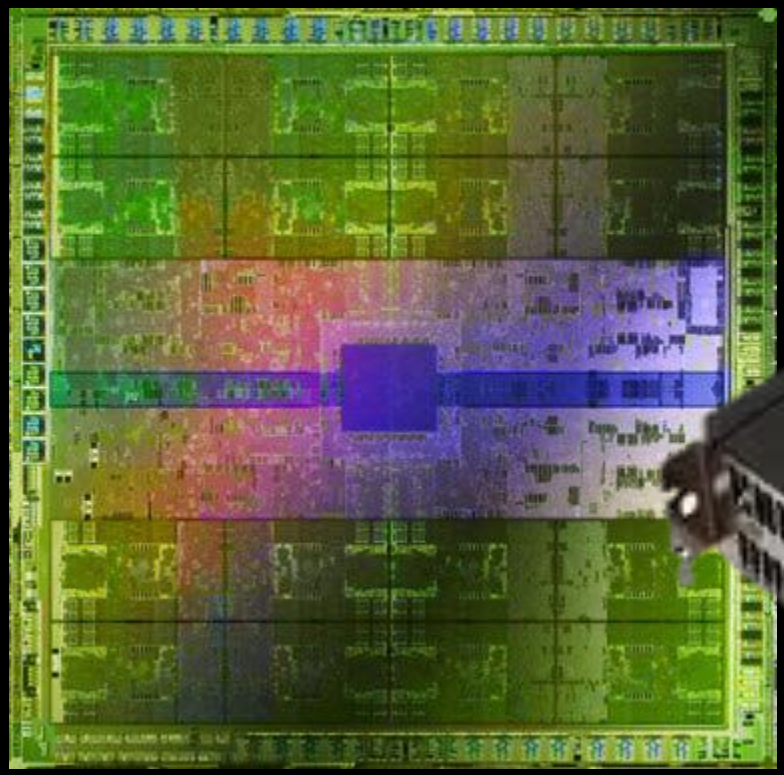
- 2004 : High level Shader language (Brook , Sh)
- 2006 : Parallel graphic processing

  Nvidia CUDA


```
#include "stdafx.h"

#include <stdio.h>
#include <cuda.h>

// Kernel that executes on the CUDA de
__global__ void square_array(float *a,
{
  int idx = blockIdx.x * blockDim.x + t
  if (idx<N) a[idx] = a[idx] * a[idx];
}

// main routine that executes on the hos
int main(void)
{
  float *a_h, *a_d;  // Pointer to host
  const int N = 10;  // Number of elemen
  size_t size
```

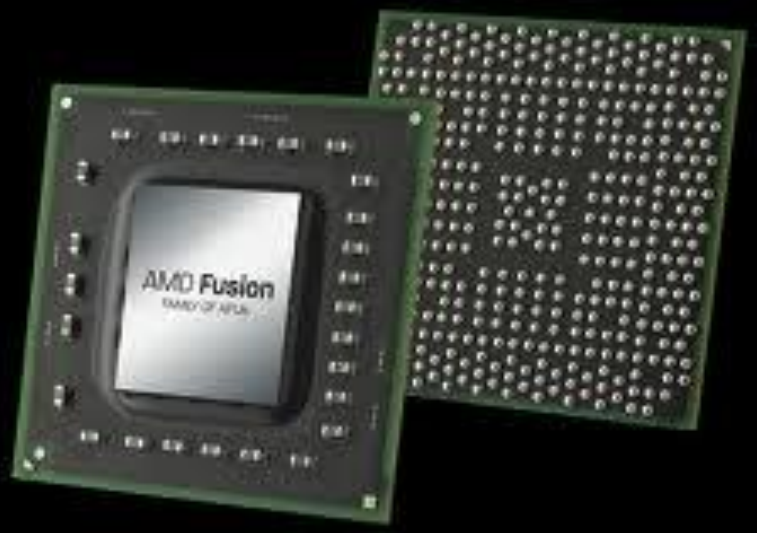
A history of GPU (7)

- 2010 : Nvidia Fermi → General purpose GPU



A history of GPU (8)

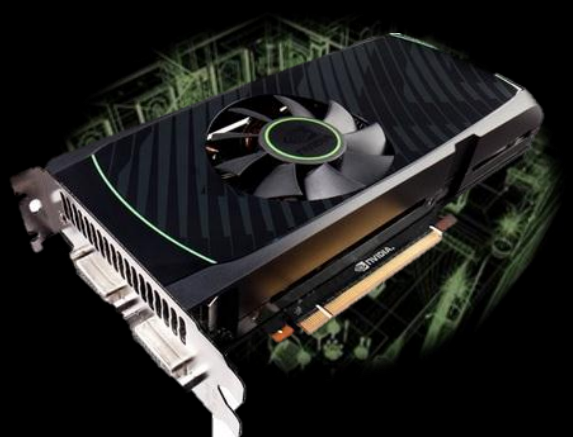
- 2010 : Combining CPU with GPU
  - AMD APU
  - Intel Larrabee



A history of GPU (9)

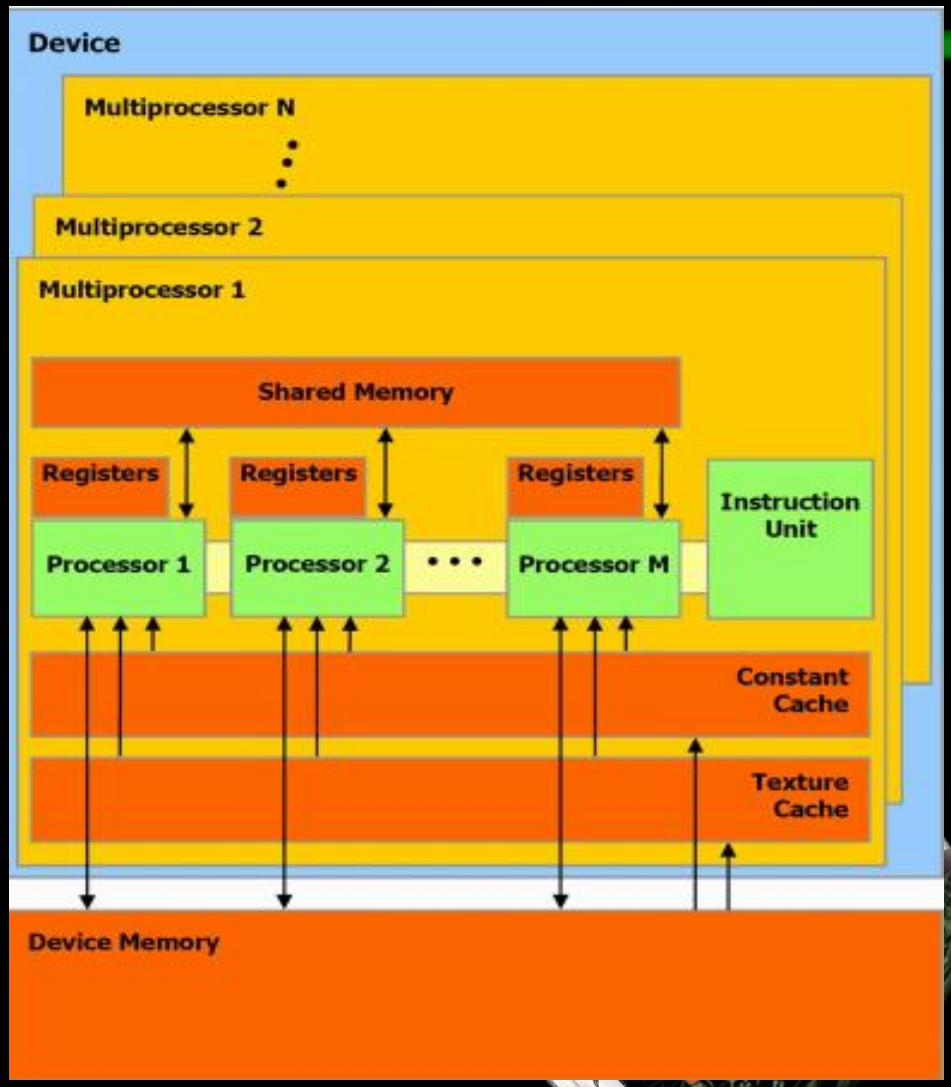
- Late 2011: GPU for multiplatform

  OpenCL



How does it work ?

- Multiprocessors
  - Multicores
- Memory level
  - Register
  - shared memory
  - constant cache
  - texture cache
  - global memory



Memory Level

- Global Memory
  for every thread
- Shared Memory
  for thread in same block
- Constant Memory
  read-only, global memory's cache

Memory Level (2)

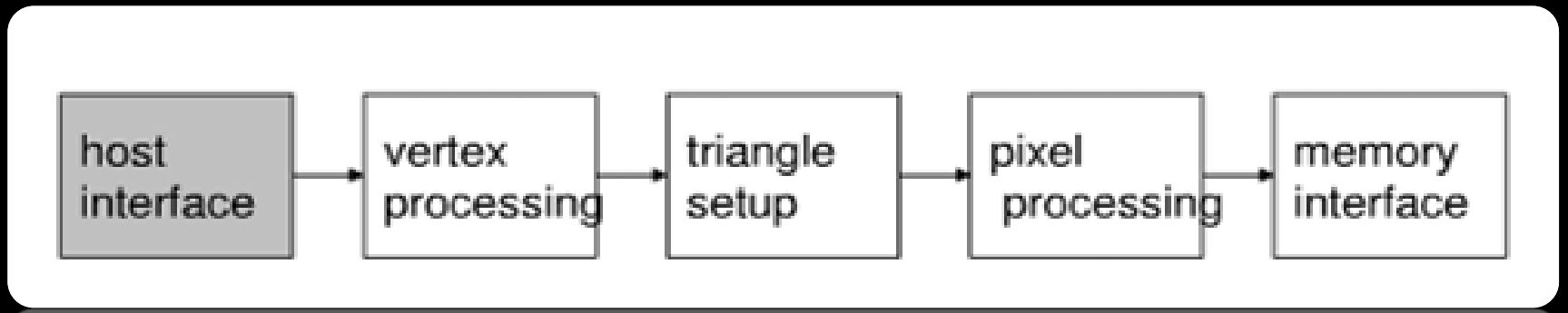
- Texture Memory
  optimized for 2D spatial locality
- Register Memory
  Owned by every thread

Main Technique

- Multiple parallel simple processors
- Multiple ALUs on each core processors
- Interleaving operation to avoid latency

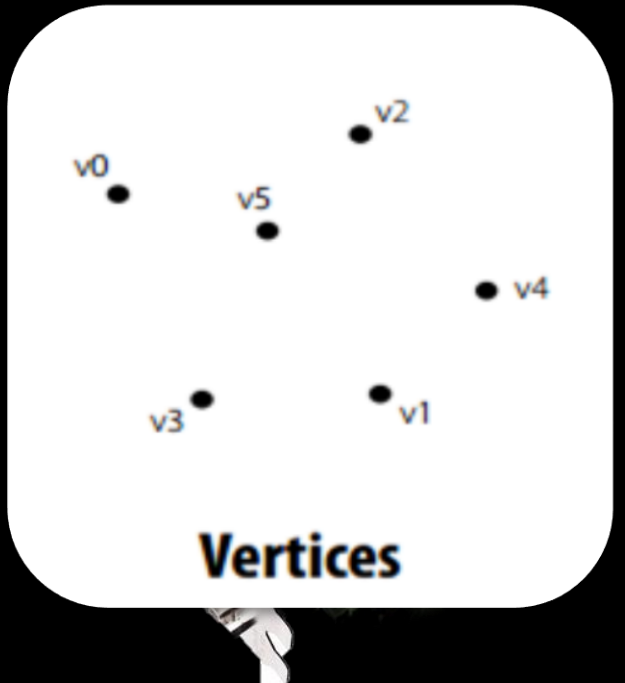
GPU processing pipeline

- Basic pipeline component of GPU



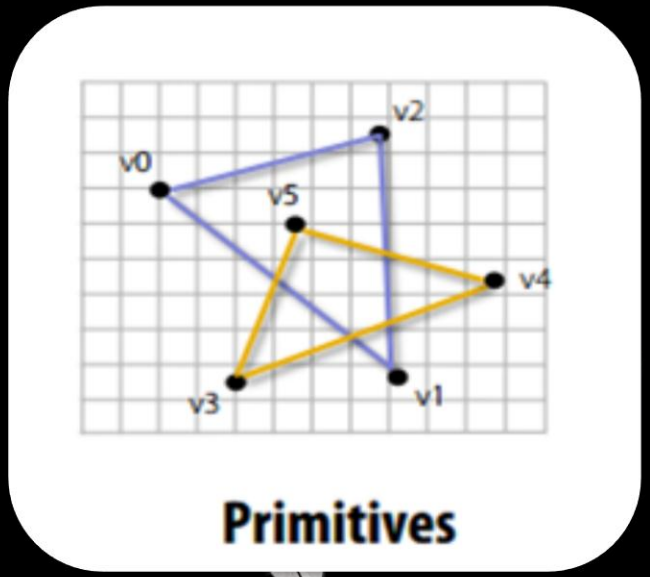
Host interface

- Main gateway between CPU and GPU
- Receive instruction and geometry information
- Returns output in form of vertex stream



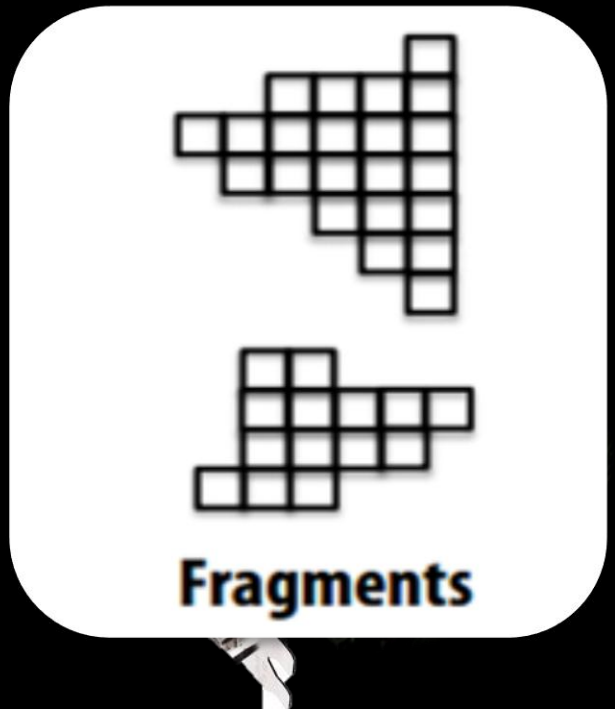
Vertex processing

- Receive vertex stream
- Mapping vertex to screen space
  - Done with linear transformation
  - Or other complex transformation (e.g : morphing)
- Does not form or remove any vertex



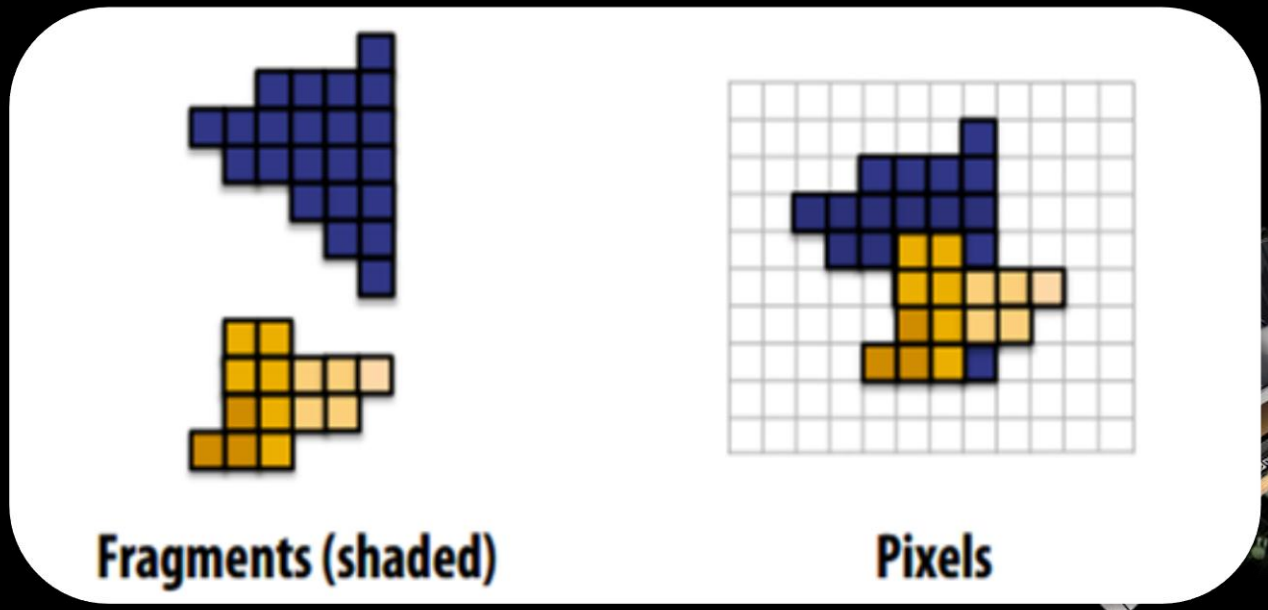
Triangle setup

- Transform screen space to pixel
- Hidden surface removal
- Fragment construction



Pixel processing

- Process data on each fragment
- Calculate the value of pixels
- Texture mapping and other arithmetic operation is done in this stage

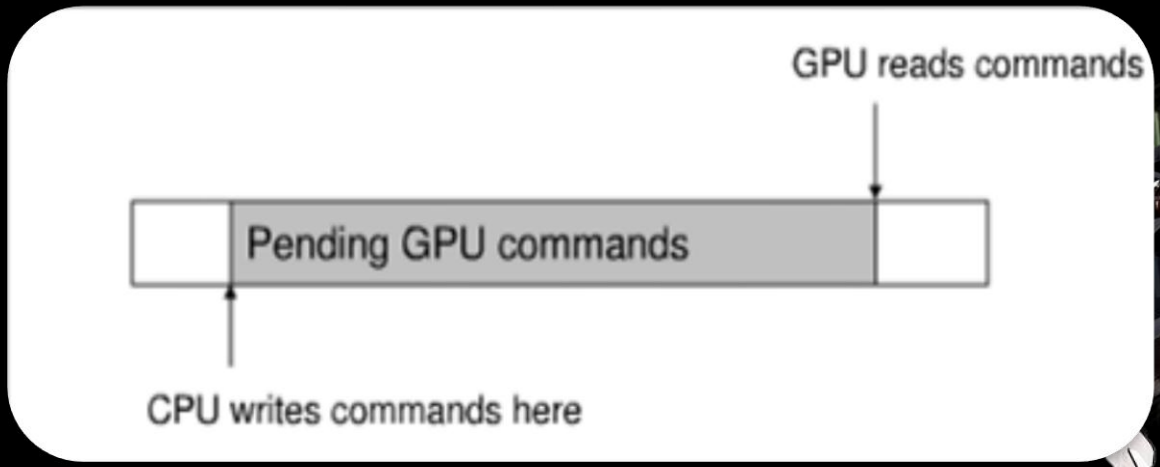


Memory interface

- Write the pixel result into the framebuffer
- Zbuffer test, stencil, and alpha test is done within this stage
- On modern GPU z and color information is compressed

Command buffer

- CPU and GPU works in parallel with each other
- CPU and GPU communicate via the Command buffer
- CPU puts instruction and GPU runs them

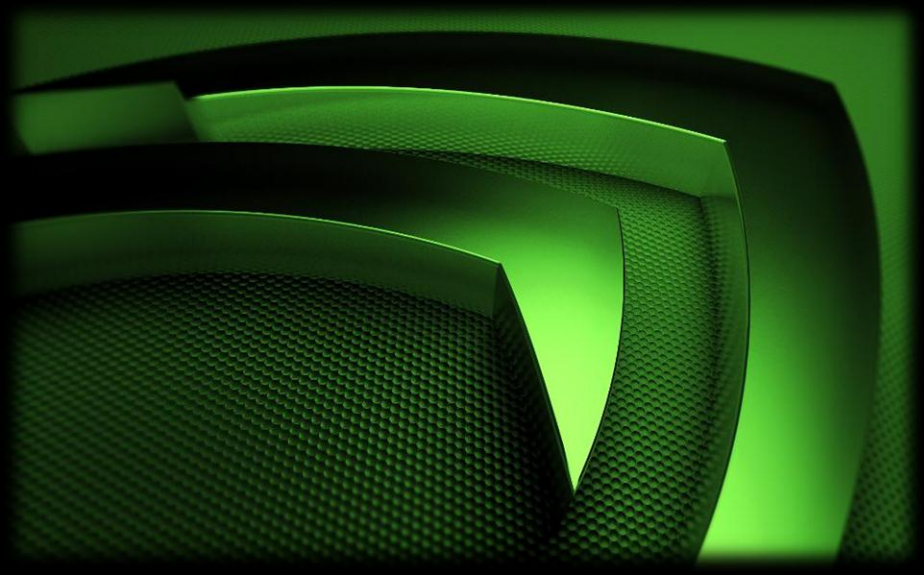


The need for synchronization

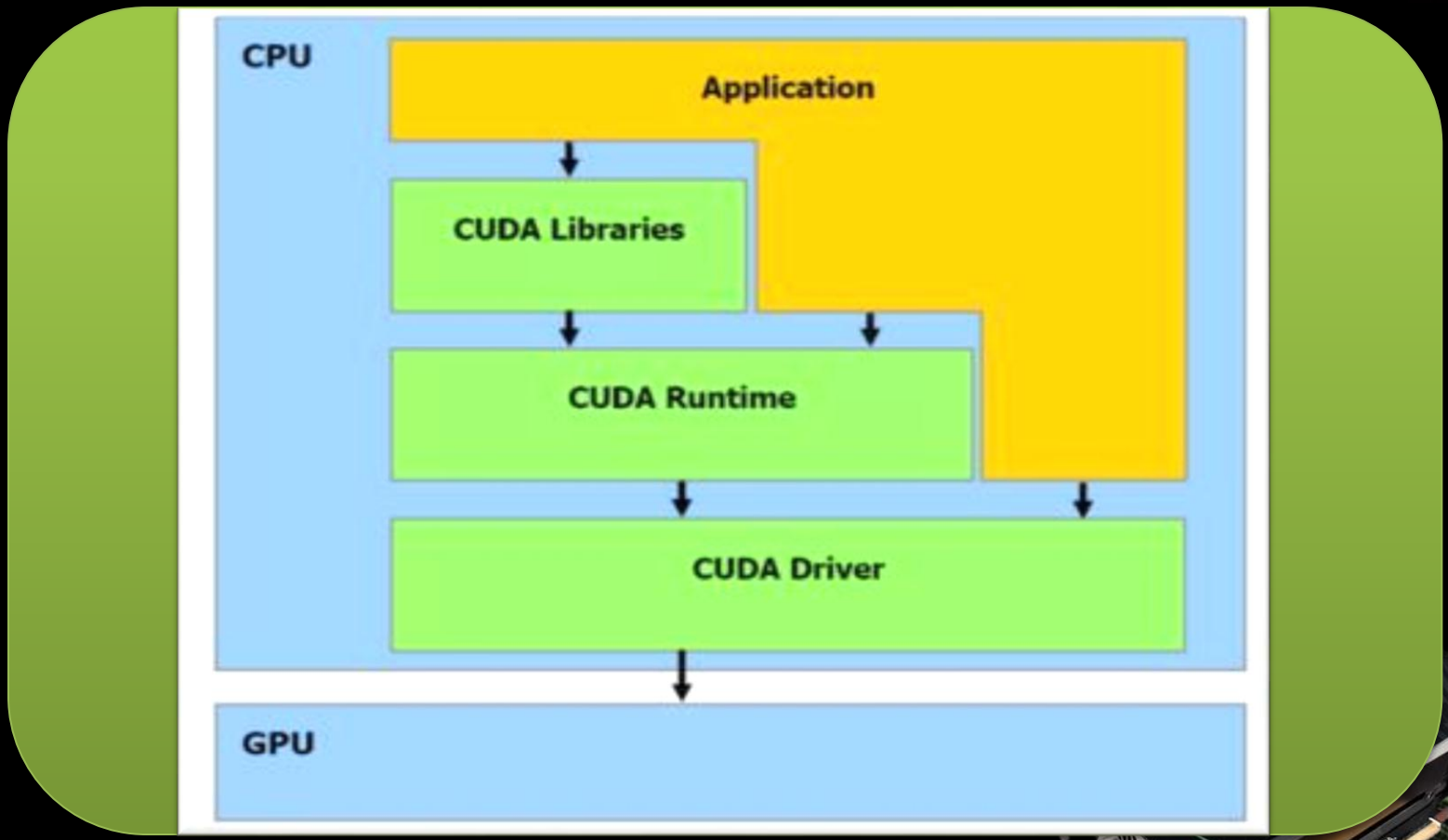
- The use of Semaphore
  - Blocks data used by GPU
  - CPU must wait resulting stall
- Inlining data
  - Data is passed within the command buffer
- Renaming data
  - Creating new block of data for temporary storage
  - Deletes them when GPU finishes

We believe in CUDA

- High level shader programming language
- Created for general purpose computing with GPU
- Applied with C, C++, Fortran

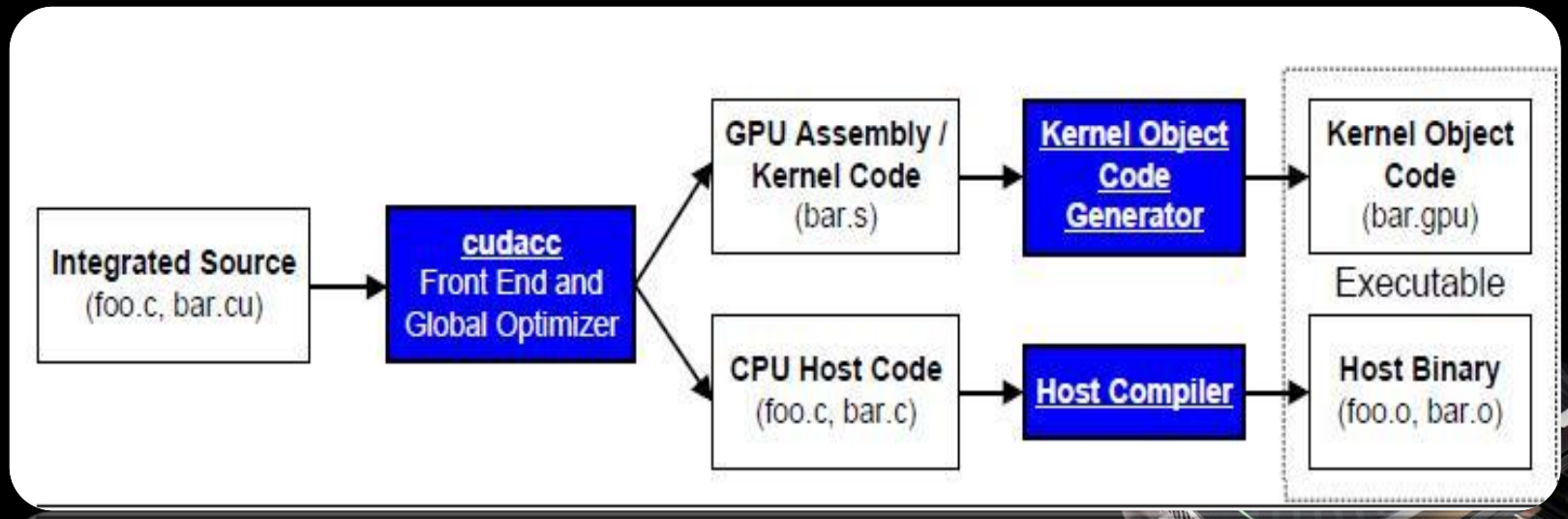


CUDA architecture



CUDA's API

- Two types of API :
  - CUDA high level API
  - CUDA low level API

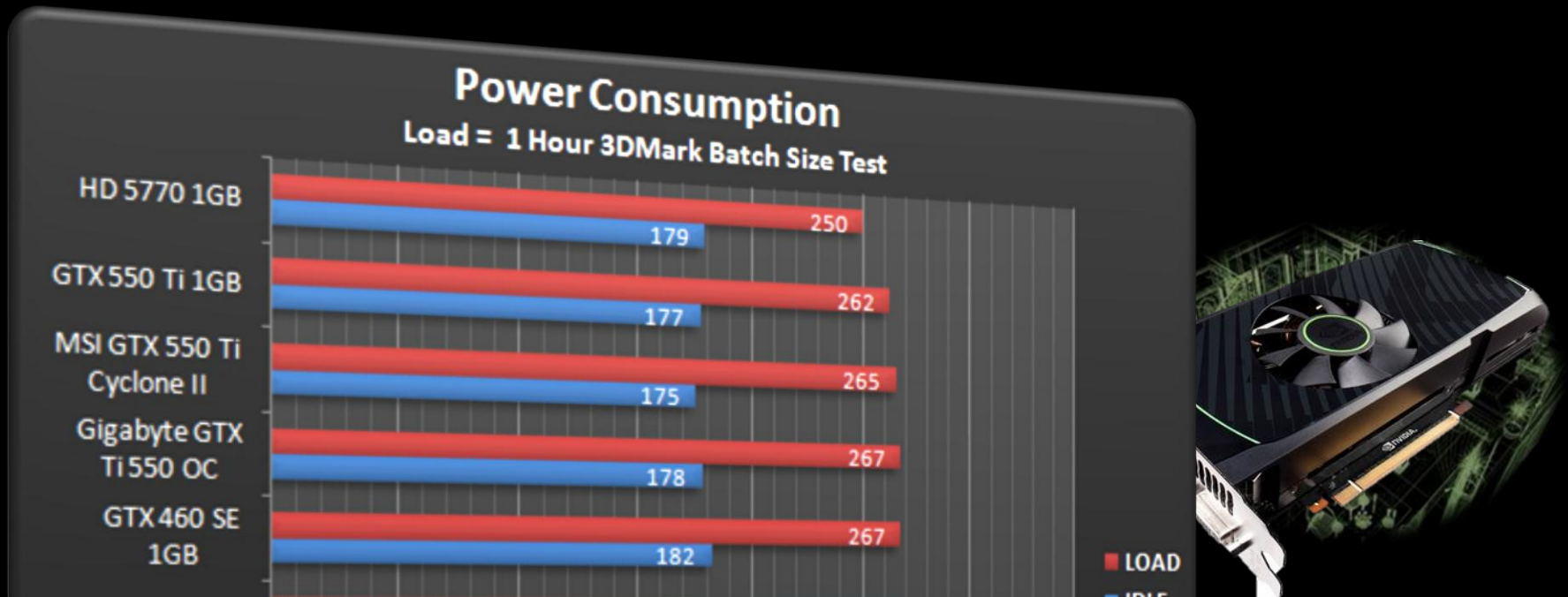


CUDA's limitation

- Low precision double floating point operation
- CUDA can only run on Nvidia's graphic card
- Does not support recursive
- Bus latency between GPU and CPU

GPU's limitation

- Low Double floating point precision
- Latency between GPU and CPU
- Large Power consumption




Thank you