

# Đồ họa



## Tuần 1

Giảng viên: Trần Đức Minh

# Nội dung bài giảng



- Đồ họa máy tính là gì ?
- Các ứng dụng đồ họa máy tính
- Hệ thống đồ họa máy tính
- Tạo lập hình ảnh
- Kiến trúc đường ống đồ họa
- Lập trình đồ họa
- WebGL



# Đồ họa máy tính là gì ?



- Đồ họa máy tính giải quyết tất cả những khía cạnh của việc **tạo ra hình ảnh bằng máy tính**.
- Đồ họa máy tính là sự kết hợp giữa 3 thành phần
  - Ứng dụng: Các đối tượng được sáng tạo bởi người dùng (đồ vật, nhân vật trong games, ...)
  - Phần mềm:
    - Mức cao: Các phần mềm được dùng để tạo dựng hình ảnh như 3DS Studio, Lightwave, ....
    - Mức thấp: Các thư viện hỗ trợ lập trình đồ họa như DirectX, OpenGL, ...
  - Phần cứng: Máy tính và card đồ họa dùng để lưu trữ và hiển thị các đối tượng đồ họa.

# Các ứng dụng đồ họa máy tính



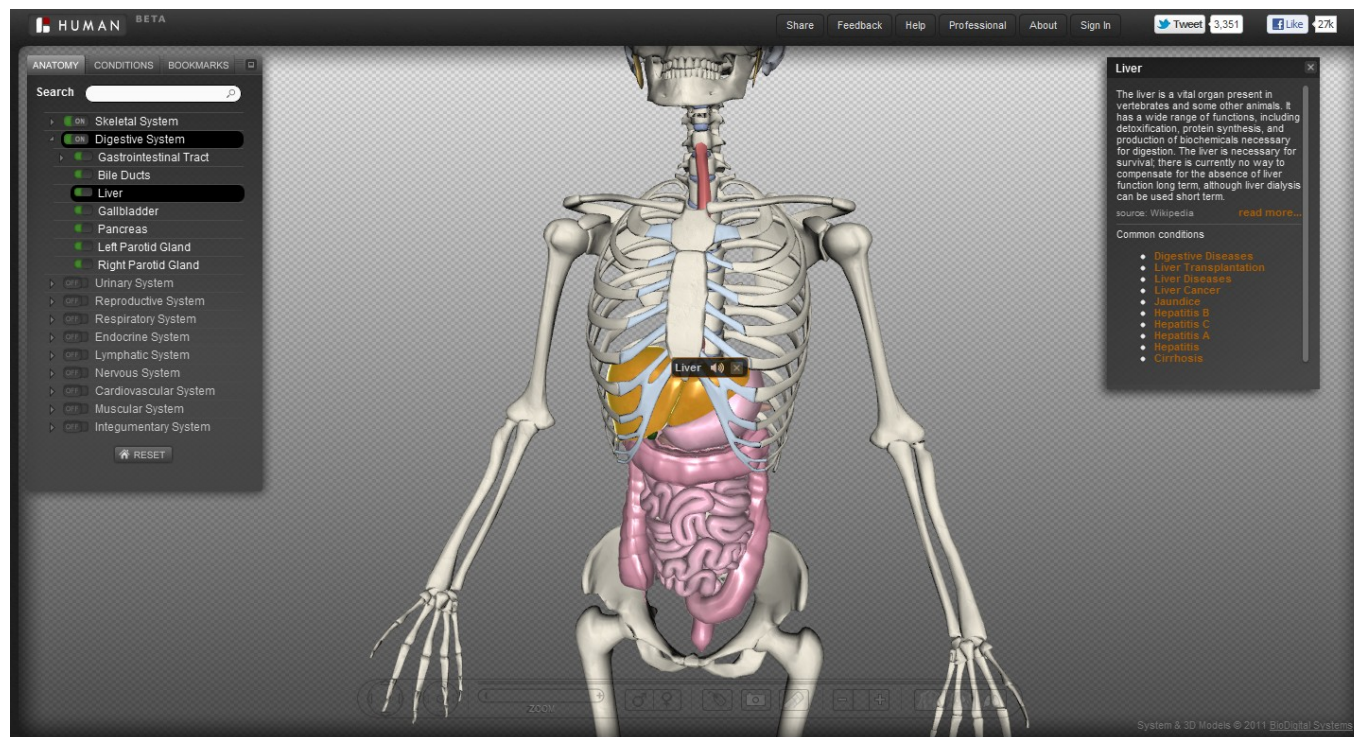
- Biểu diễn thông tin
  - Đồ thị, mặt cong thay cho biểu bảng
  - Biểu diễn khác
    - Ví dụ:  
<https://artsexperiments.withgoogle.com/freefall/wave>



# Các ứng dụng đồ họa máy tính



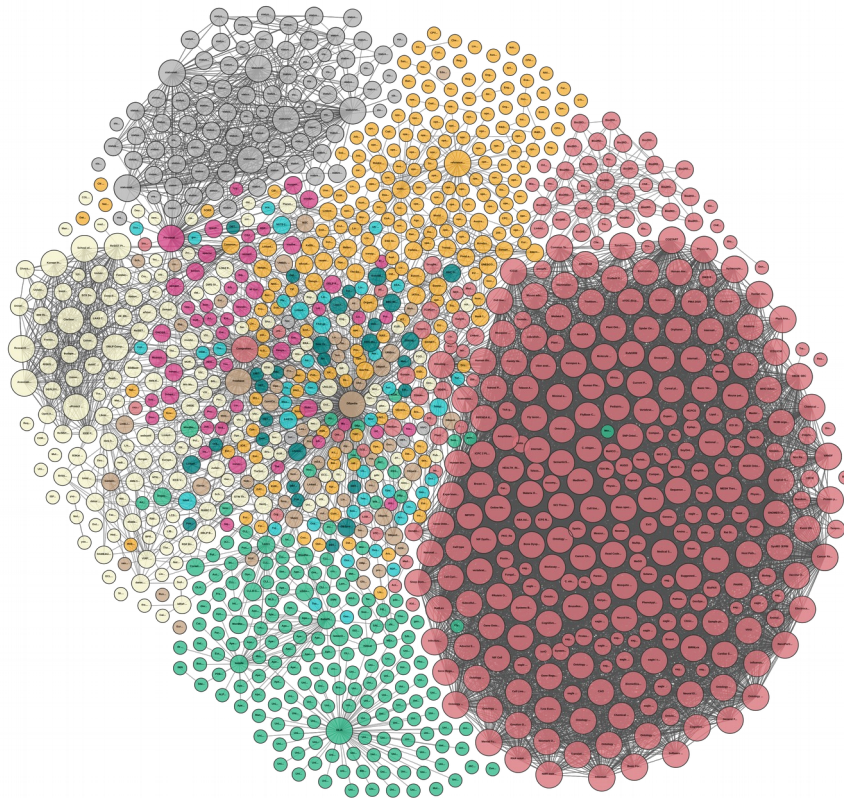
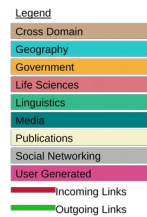
- Hiện thị thông tin
  - Hình ảnh giải phẫu cơ thể người



# Các ứng dụng đồ họa máy tính



- Hiện thị thông tin
  - Linked Open Data

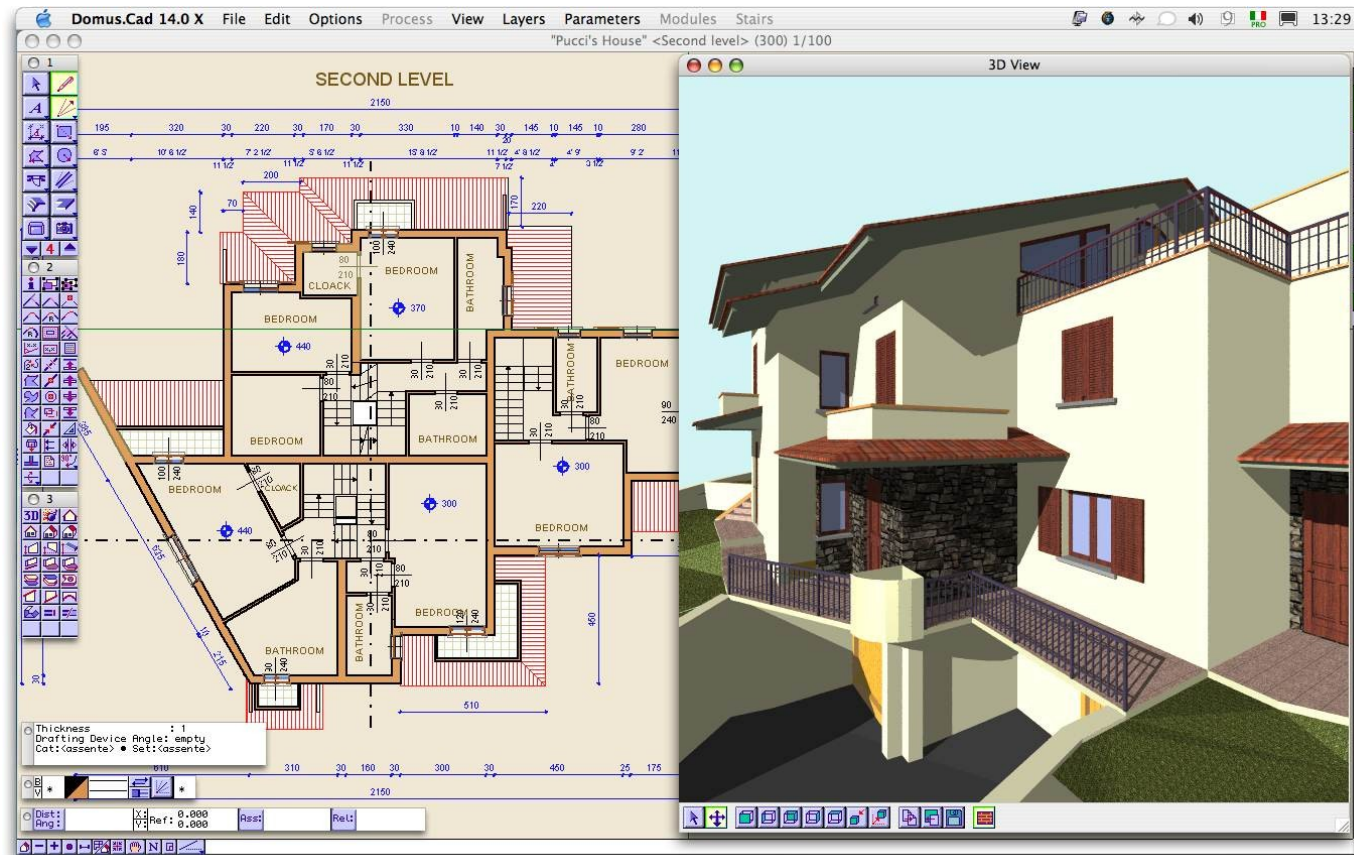




# Các ứng dụng đồ họa máy tính



- Máy tính hỗ trợ thiết kế



# Các ứng dụng đồ họa máy tính



- Các hệ thống giả lập



Máy bay



Tramway



# Các ứng dụng đồ họa máy tính



- Thực tại ảo



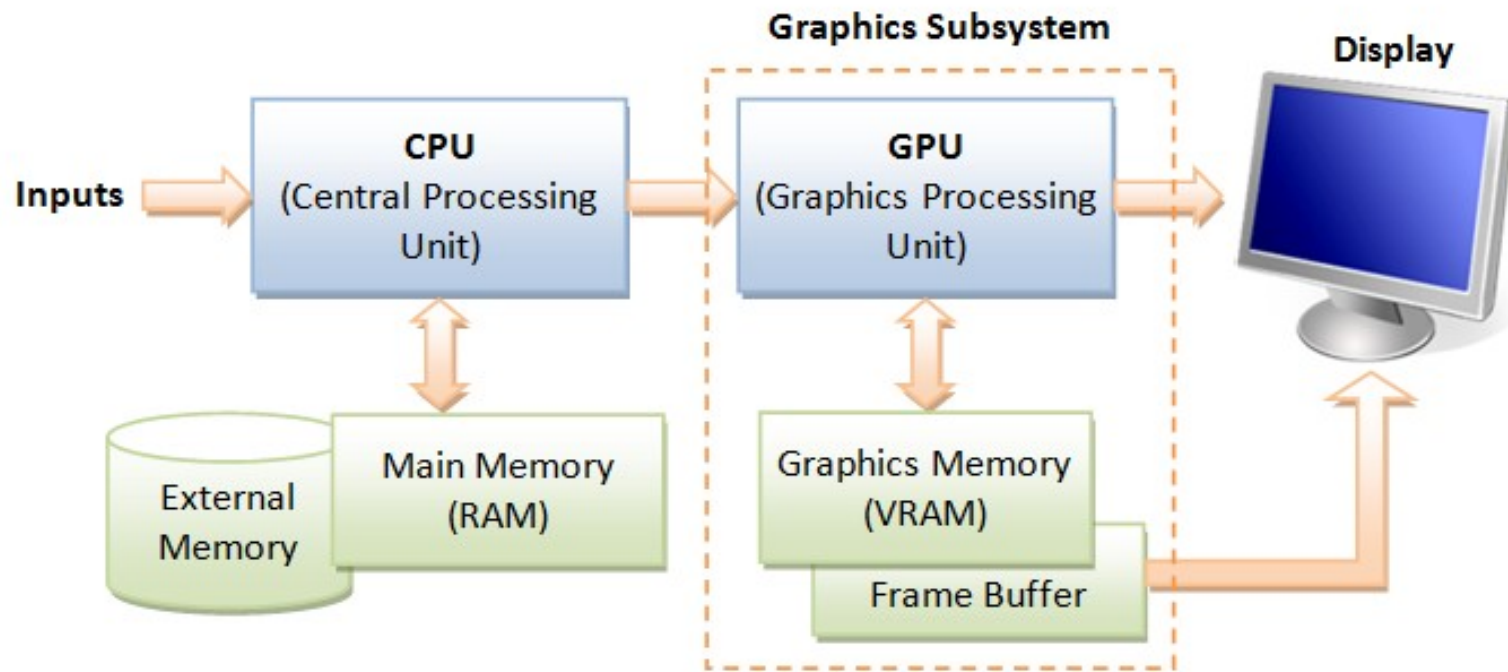
# Các ứng dụng đồ họa máy tính



- Augmented reality (thực tế tăng cường)



# Hệ thống đồ họa máy tính

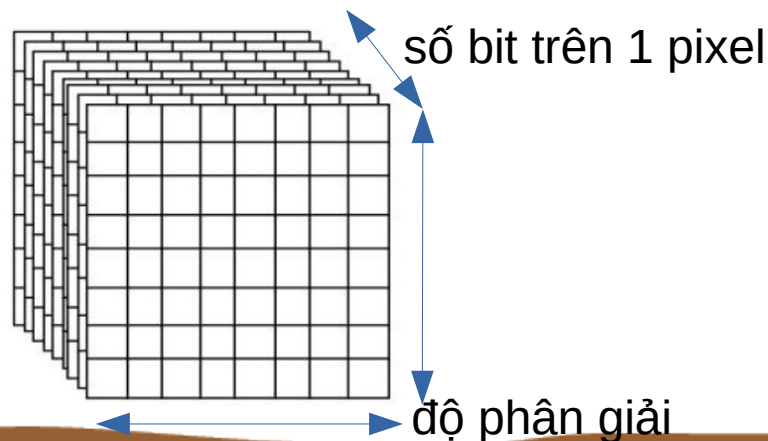


Hình ảnh trong Frame buffer là hình ảnh hiển thị lên màn hình

# Frame Buffer



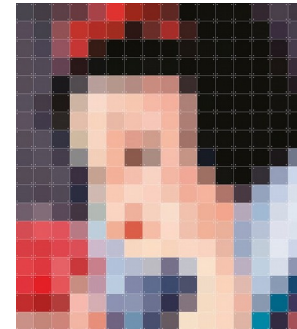
- Nằm trong **bộ nhớ máy tính** hoặc bên trong **GPU** (Graphics Processing Unit).
- Được sử dụng để lưu trữ thông tin các điểm ảnh (pixel) trên màn hình máy tính.
- Độ phân giải của Frame Buffer và độ phân giải của màn hình là giống nhau.



# Pixel



- Pixel là đơn vị nhỏ nhất của ảnh, trong đó ảnh là một mảng 2 chiều của các điểm ảnh.
- Mỗi pixel có 2 thuộc tính
  - Vị trí của pixel - Tọa độ x và y.
  - Giá trị của pixel - Màu của pixel
    - Số lượng bit được sử dụng cho mỗi pixel sẽ xác định số lượng màu có thể được tạo ra cho ảnh.
    - Ví dụ:
      - 1 bit : Số lượng màu của ảnh là 2
      - 8 bits : Số lượng màu của ảnh là 256

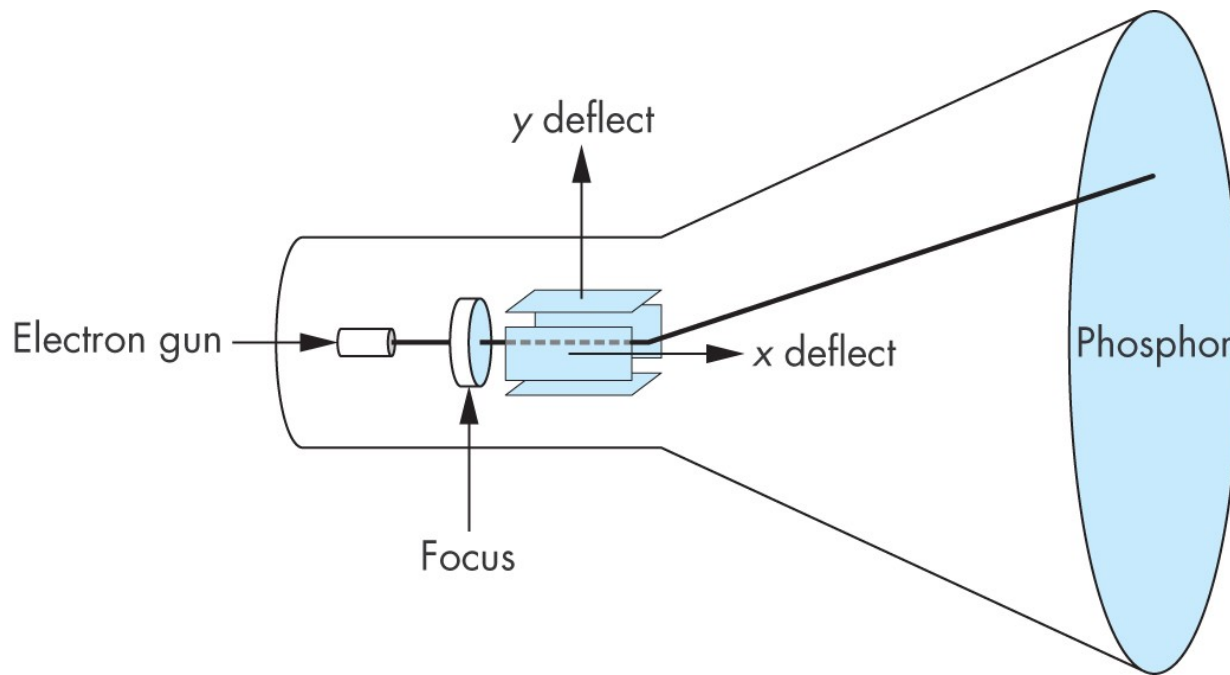




# Thiết bị xuất



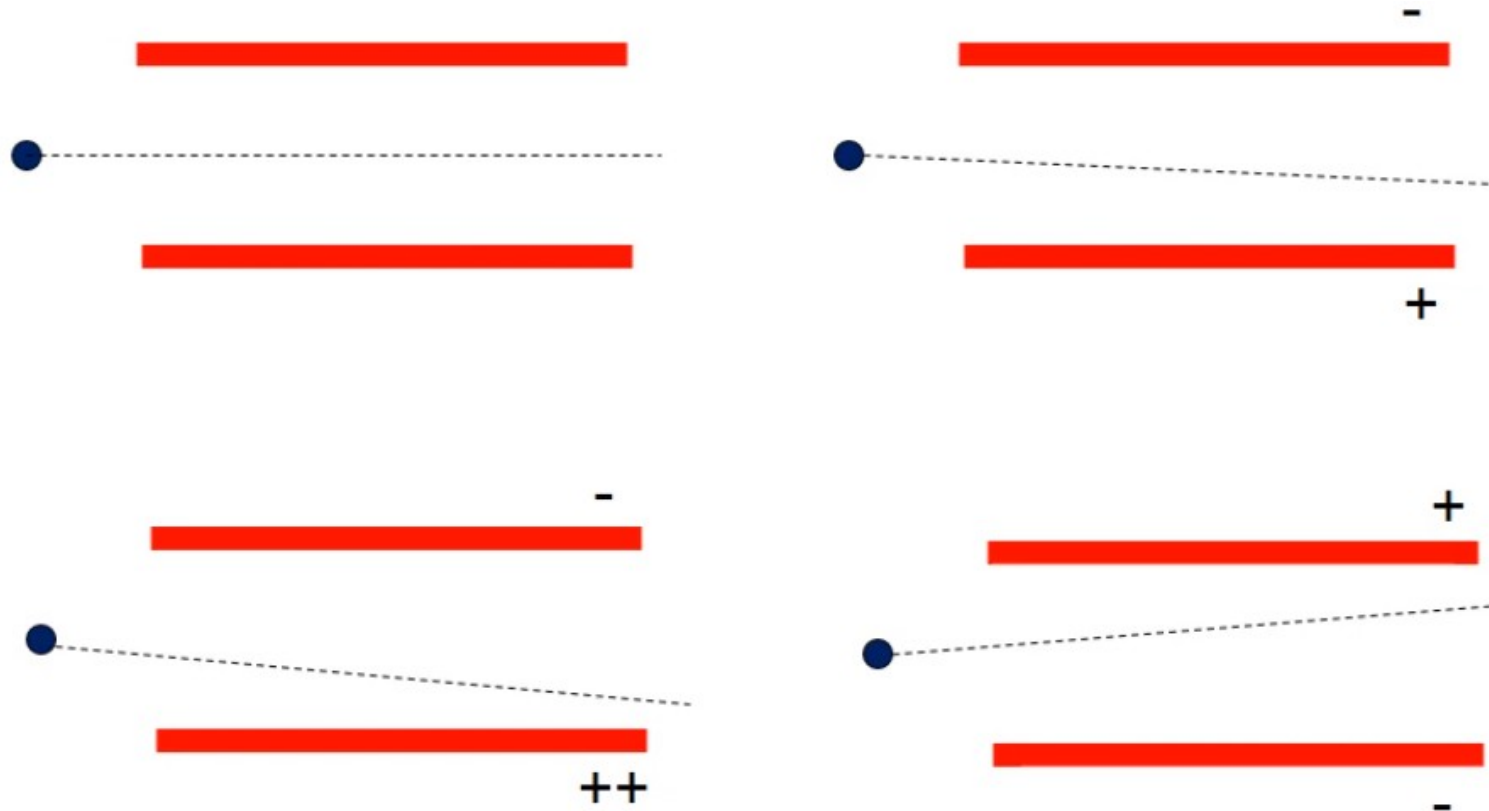
- Cathode Ray Tube (CRT)



# Thiết bị xuất



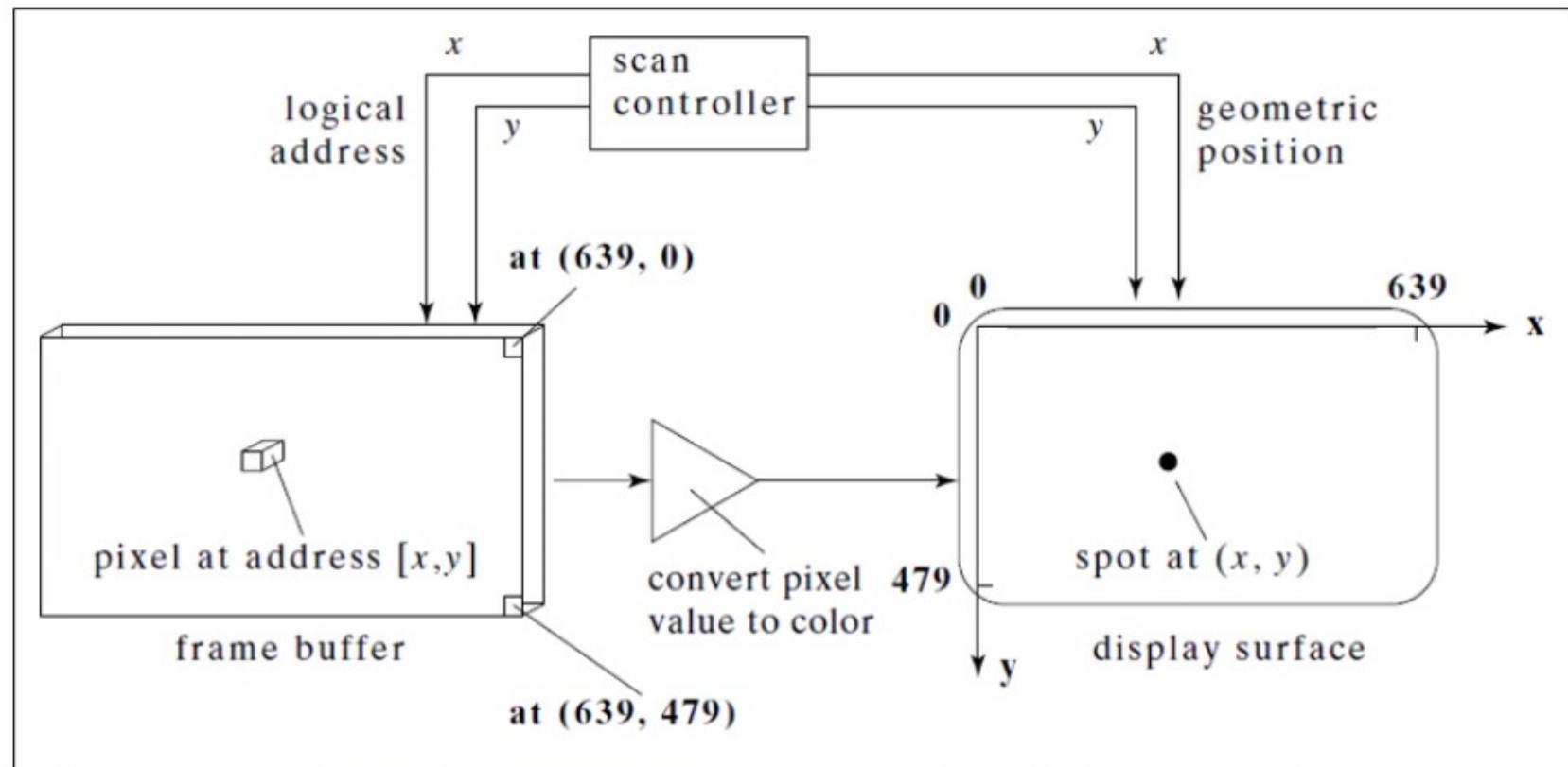
- Cathode Ray Tube (CRT)



# Thiết bị xuất



- Cathode Ray Tube (CRT)



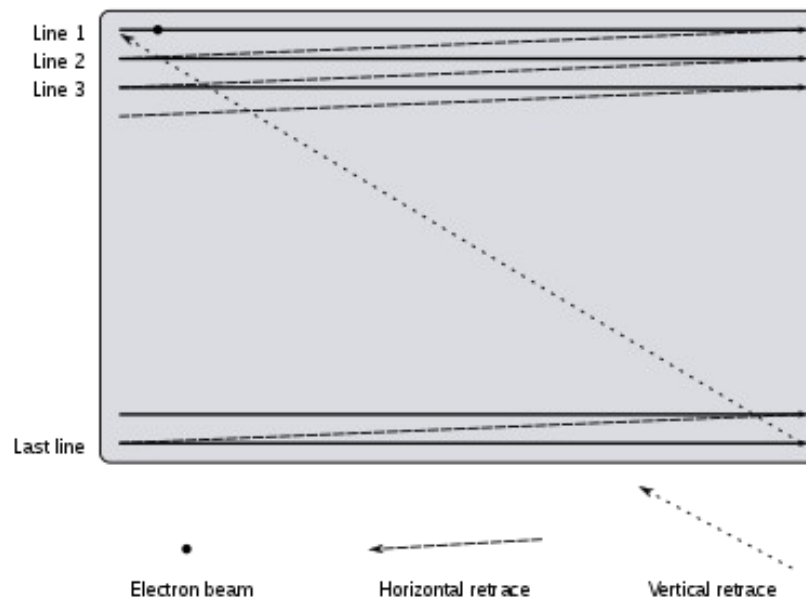
# Thiết bị xuất



- Cathode Ray Tube (CRT)

- **Scan controller:**

- Tần số làm tươi:  $x\text{Hz}$  = Trong 1 giây làm tươi  $x$  lần (Mỗi 1 giây thì cần  $x$  lần đọc dữ liệu từ Frame Buffer để đưa lên màn hình)



# Thiết bị xuất

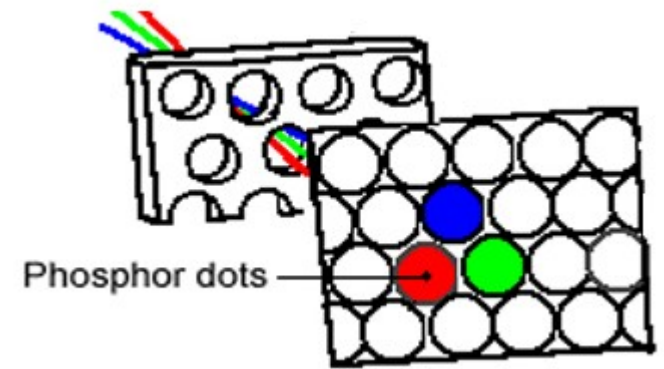


- Cathode Ray Tube (CRT)
  - **Convert pixel:** Chuyển đổi giá trị trong Frame Buffer thành màu sắc hiển thị trên màn hình.





# Thiết bị xuất

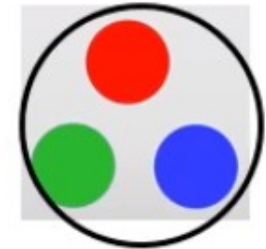
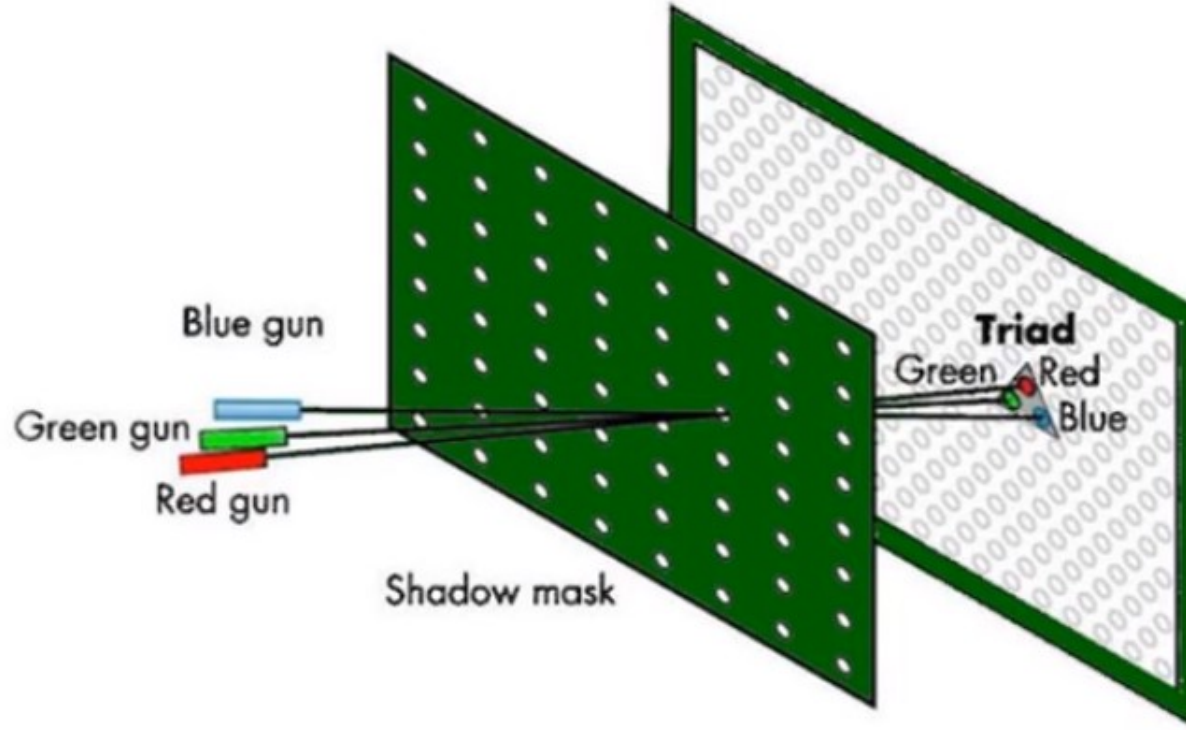


- Màn hình CRT màu
  - Mỗi pixel trên màn hình chứa 3 chấm, tương ứng với 3 màu **Red**, **Green** và **Blue**.
  - Scan controller sử dụng 3 súng điện tử.
  - Mỗi súng điện tử có nhiều mức độ bắn khác nhau. Tương ứng với mỗi mức độ, sẽ hiển thị cường độ màu tương ứng đối với chấm.

# Thiết bị xuất



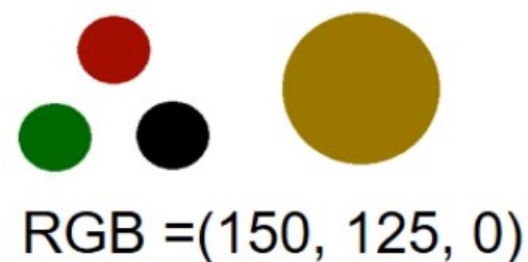
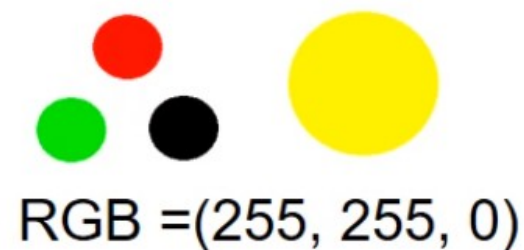
- Màn hình CRT màu



# Thiết bị xuất



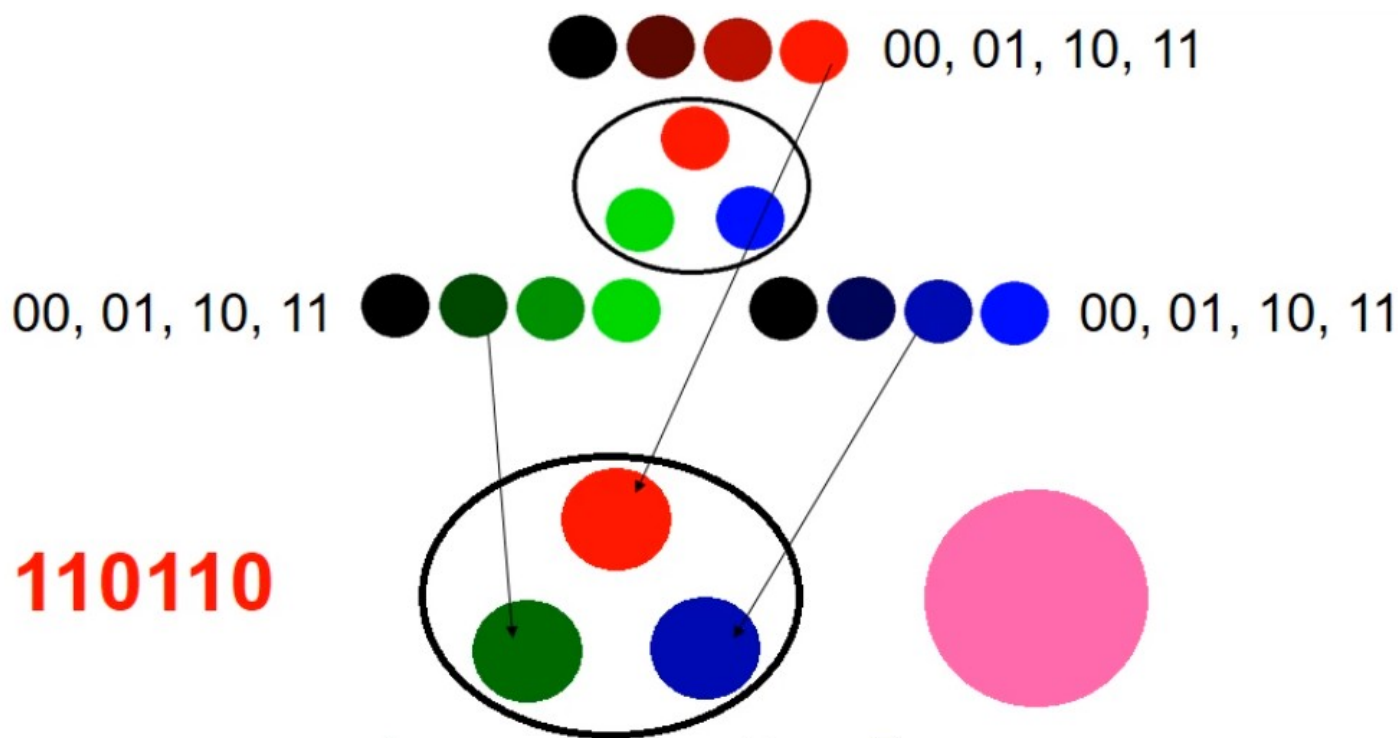
- Màn hình CRT màu



# Thiết bị xuất



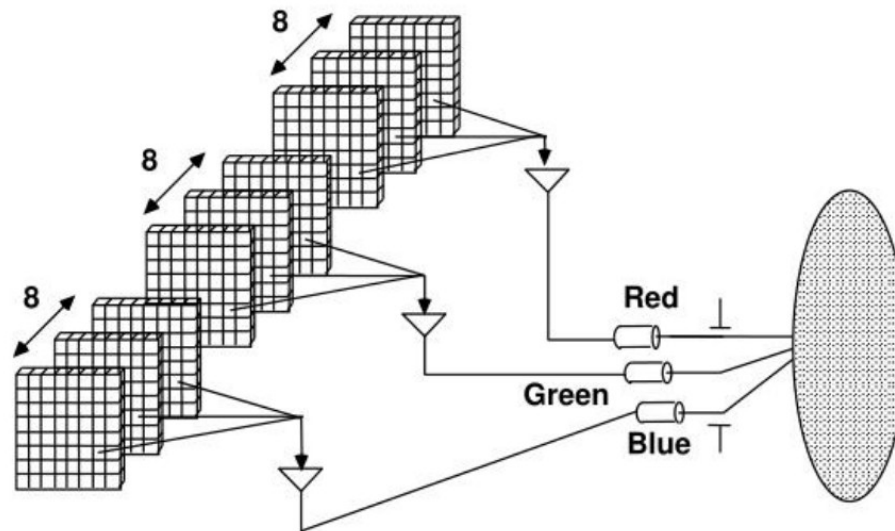
- Ví dụ: Sử dụng 6 bit để lưu trữ màu



# Thiết bị xuất



- Khi số lượng màu có thể hiển thị trên màn hình nhỏ hơn hoặc bằng số lượng màu frame buffer có thể quản lý
  - Sử dụng màu trực tiếp



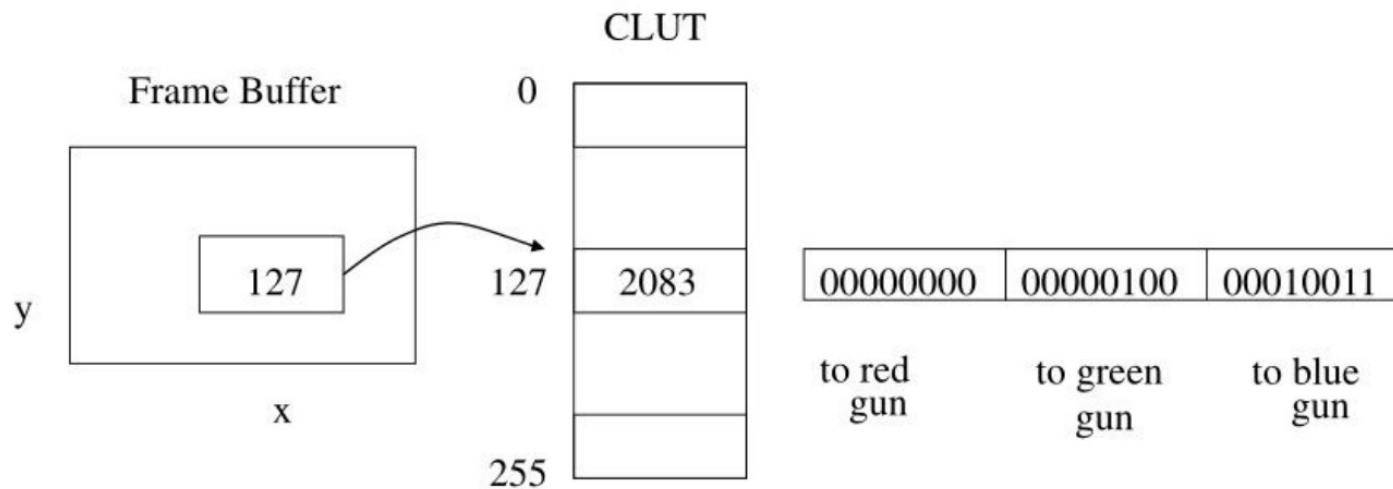
24 bits màu, chia ra 8 bit cho một súng (mỗi súng có 256 mức độ)



# Thiết bị xuất



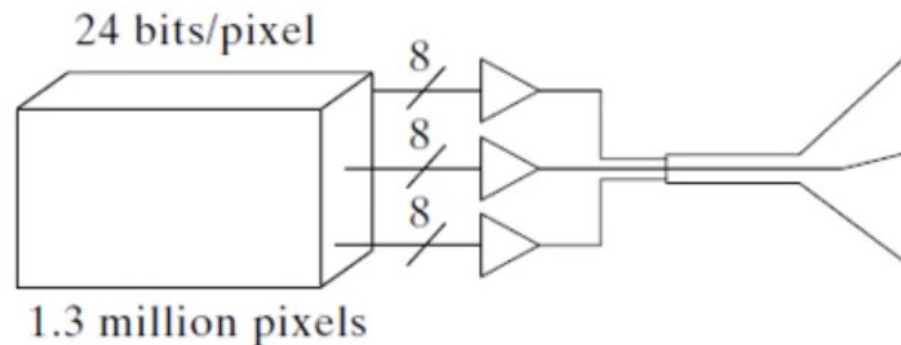
- Khi số lượng màu có thể hiển thị trên màn hình nhiều hơn số lượng màu frame buffer có thể quản lý
  - Sử dụng Color Look-up Table



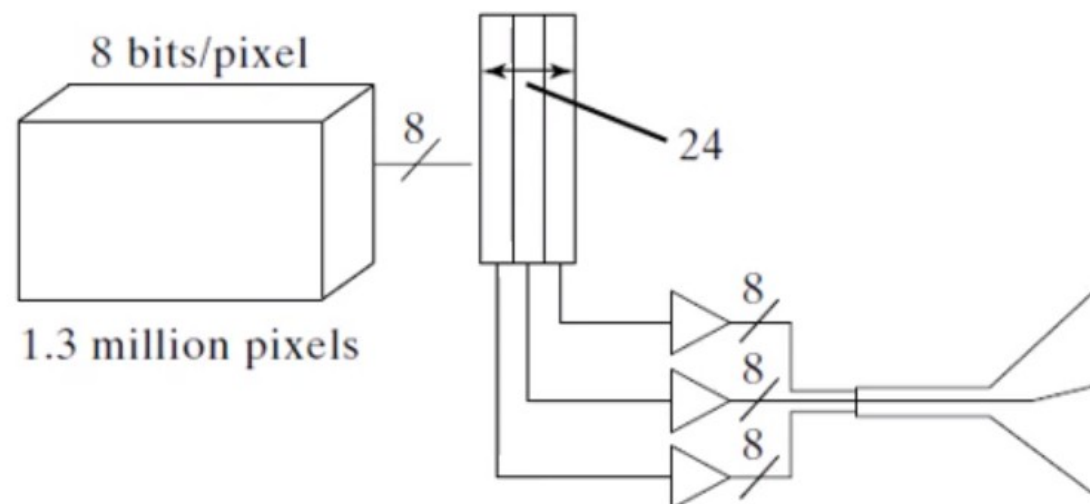
# Thiết bị xuất



expensive (frame buffer needs ~ 4Mbyte)



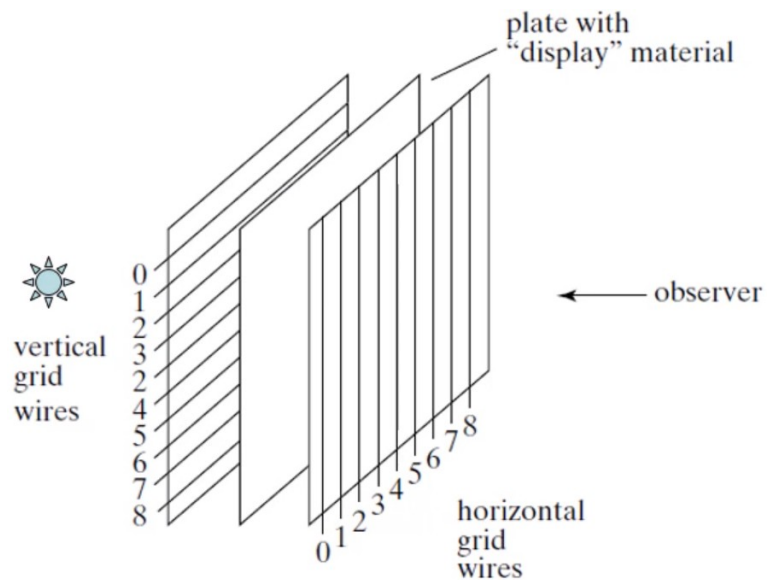
inexpensive (frame buffer needs ~ 1Mbyte)



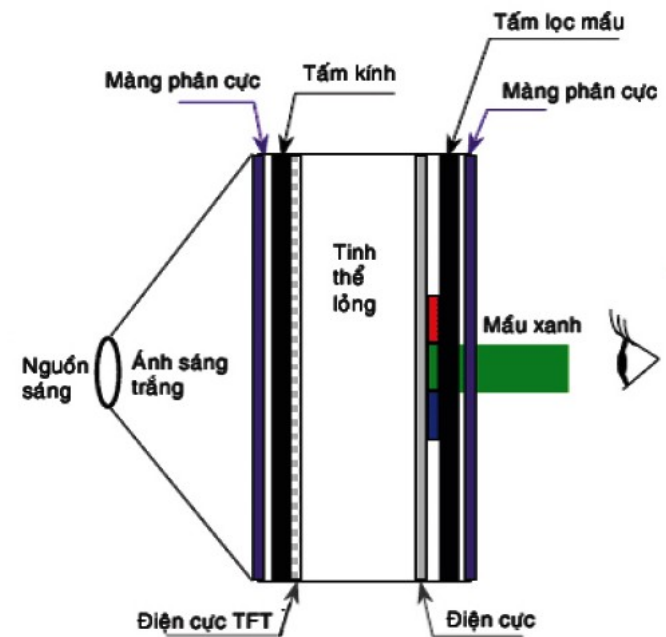
# Thiết bị xuất



- Màn hình tinh thể lỏng



Màn hình đen trắng



Màn hình màu

# Tạo lập hình ảnh



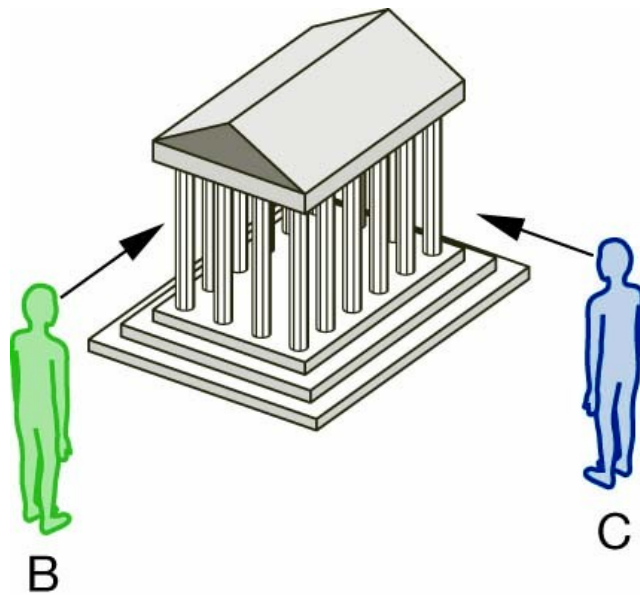
- Để tạo lập hình ảnh bằng thư viện đồ họa, ta cần xác định các thành phần sau:
  - Các đối tượng (objects)
  - Các vật liệu (materials)
  - Góc nhìn (viewer)
  - Nguồn sáng (light source)



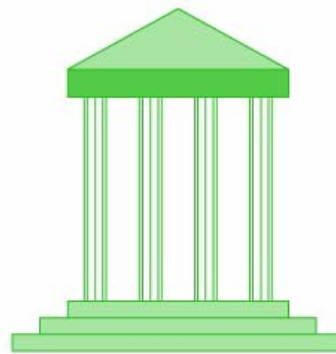
# Tạo lập hình ảnh



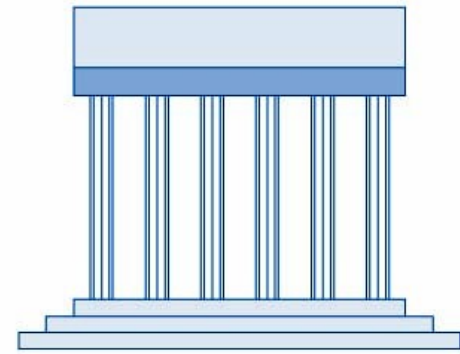
- Ví dụ đối tượng và góc nhìn



(a)



(b)



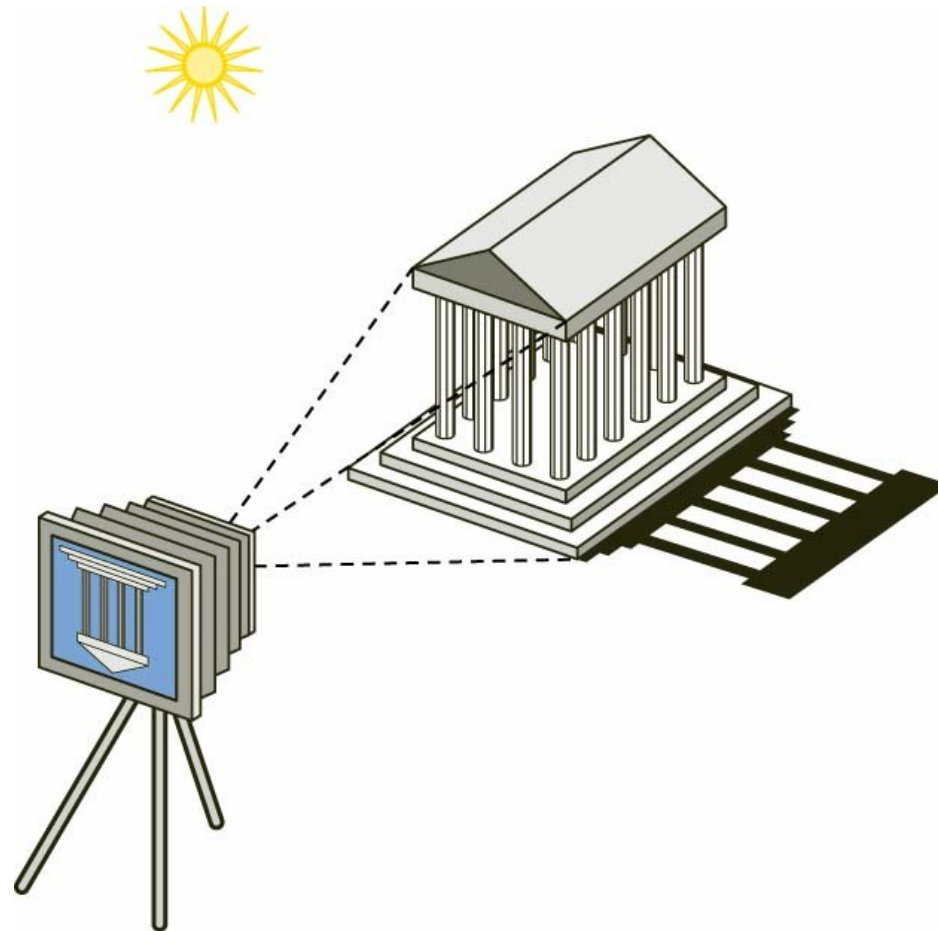
(c)



# Tạo lập hình ảnh



- Ví dụ nguồn sáng



# Kiến trúc đường ống đồ họa (pipeline)



- Kiến trúc pipeline chia xử lý ra thành nhiều khâu độc lập với nhau để đơn giản hóa.
- Tất cả các bước bên trong **đường ống đồ họa** có thể được **thực hiện bên trong phần cứng** (cụ thể là bên trong **card đồ họa**)
- Xử lý thông tin ở các đỉnh có thể **xử lý song song** để tăng tốc độ chương trình.

# Vertex processor



- **Xử lý đỉnh** (vertex processor) là chuyển đổi mô tả đối tượng từ hệ trục tọa độ này sang hệ trục tọa độ khác.
  - Tọa độ của đối tượng
  - Tọa độ của camera
  - Tọa độ của màn hình
- Tất cả việc chuyển đổi tọa độ này tương đương với việc biến đổi ma trận.
- Việc xử lý đỉnh cũng tính toán luôn cả màu sắc của đỉnh.

# Primitive Assembly



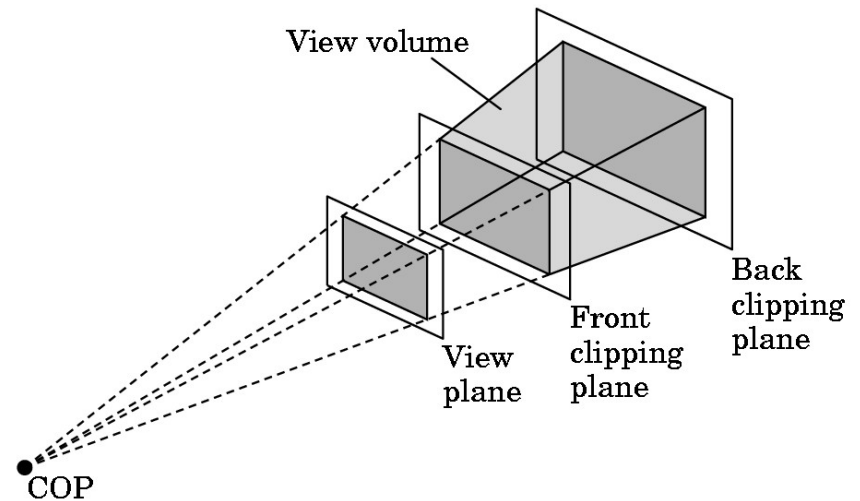
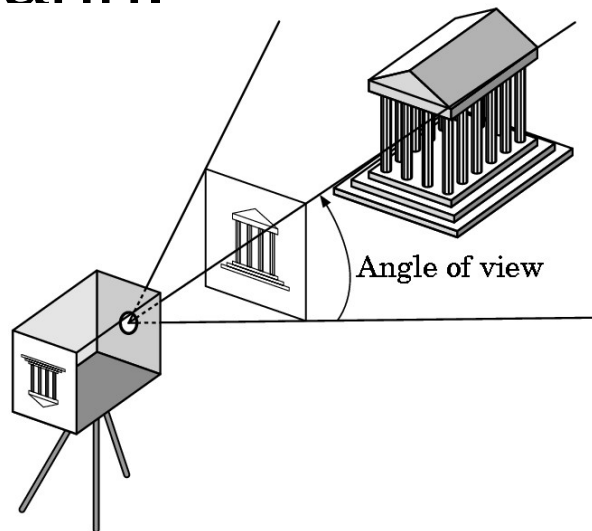
- Dựa trên các đỉnh, các đối tượng hình học sẽ được tạo ra.
  - Các đoạn thẳng
  - Các đa giác
  - Các đường cong
  - Các bề mặt



# Clipping



- Các đối tượng không nằm trong vùng **View Volume** sẽ **bị cắt xén** (clipped out) ra khỏi bối cảnh.





# Rasterization



- Rasterizer sinh ra các fragments cho mỗi đối tượng
- Fragments là các pixels mà
  - Nằm trong frame buffer
  - Có thuộc tính màu và chiều sâu (số bit màu)



# Fragment Processing



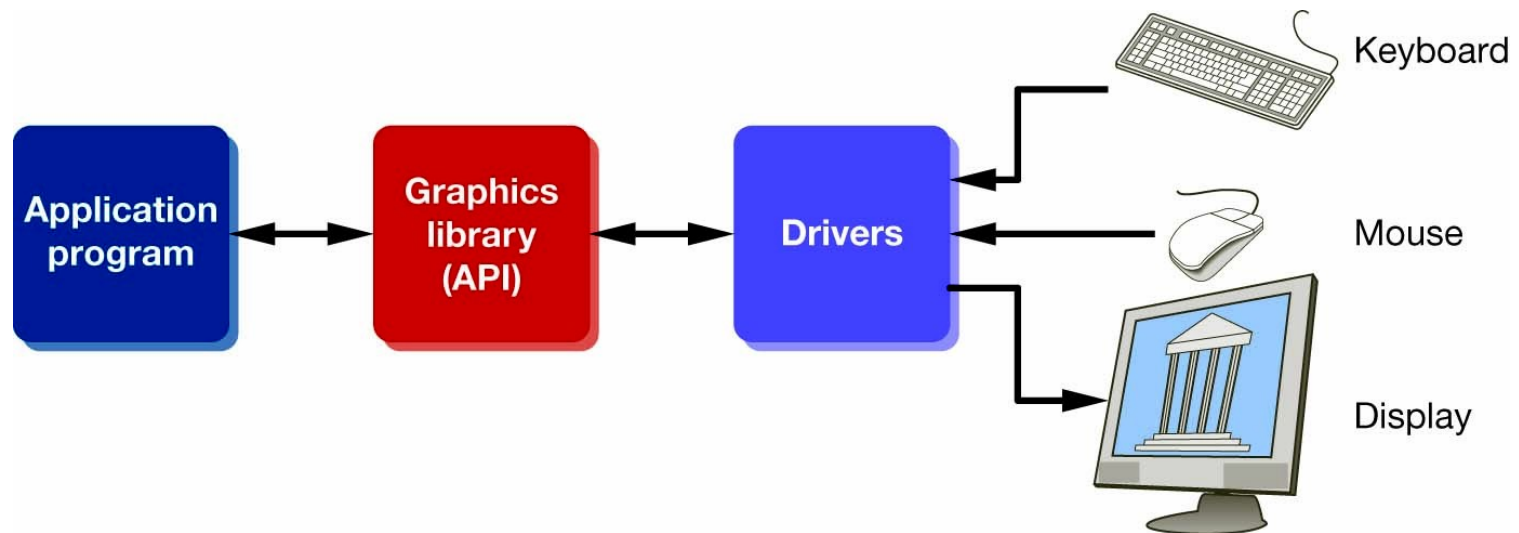
- Các fragments được xử lý để xác định màu của pixel trên màn hình tương ứng với giá trị trong frame buffer.
- Các màu được xác định bởi texture hoặc nội suy theo màu của đỉnh.



# Lập trình đồ họa



- Người lập trình sẽ lập trình đồ họa thông qua một giao tiếp phần mềm **API** (Application Programmer Interface)



# Lập trình đồ họa



- Thư viện đồ họa cung cấp cho người lập trình các hàm cần thiết để **tạo dựng và đặc tả các thành phần**
  - Các đối tượng (objects)
    - Mất thời gian nhất
  - Các vật liệu (materials)
  - Góc nhìn (viewer)
  - Nguồn sáng (light source)
- Ngoài ra, thư viện đồ họa cũng cung cấp các hàm hỗ trợ cho xử lý
  - Thiết bị đầu vào (chuột, bàn phím)
  - Các khả năng khác liên quan đến hệ thống

# Lập trình đồ họa



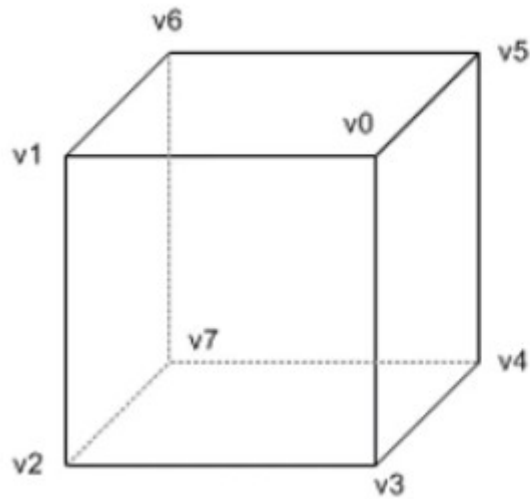
- Đặc tả đối tượng
  - Thư viện đồ họa không hỗ trợ các hàm để vẽ các đối tượng phức tạp, do đó để vẽ các đối tượng phức tạp ta cần phải tách nó thành các đối tượng đơn giản.
  - Thư viện đồ họa chỉ hỗ trợ vẽ các đối tượng sau
    - Điểm
    - Đường thẳng
    - Đa giác
    - Một số đường cong và bề mặt



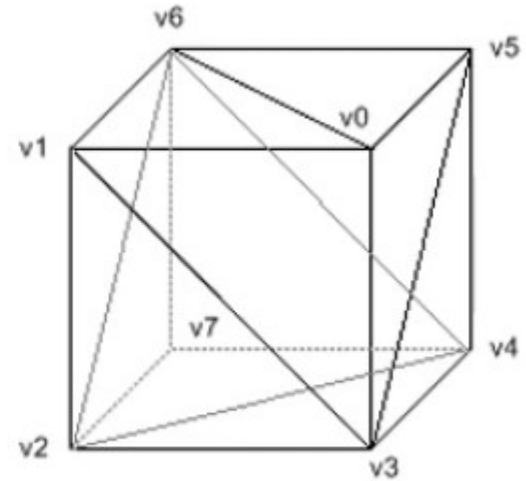
# Lập trình đồ họa



- Đặc tả đối tượng
  - Ví dụ vẽ hình lập phương



**Chia thành 6 hình vuông**

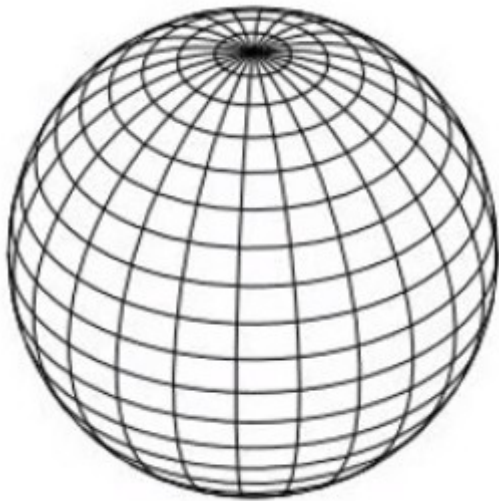


**Chia thành 12 hình tam giác**

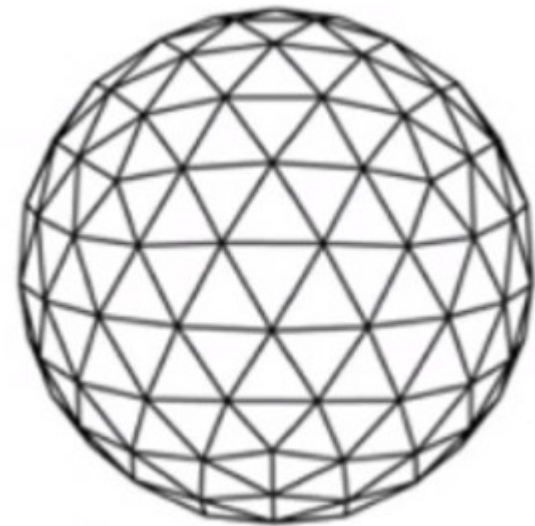
# Lập trình đồ họa



- Đặc tả đối tượng
  - Ví dụ vẽ hình cầu



**Chia thành các hình thang  
(chia theo kinh tuyến - vĩ tuyến)**



**Chia thành các hình tam giác**

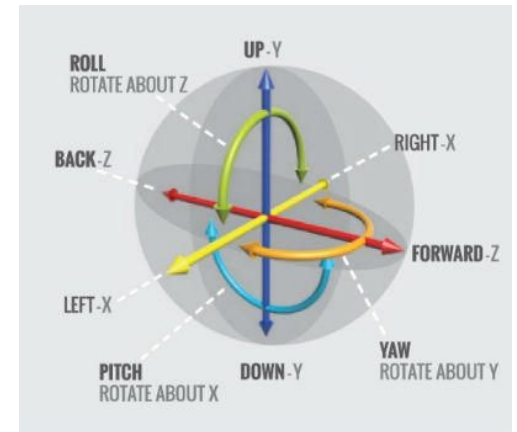




# Lập trình đồ họa



- Đặc tả góc nhìn (viewer)
  - Việc thiết lập khá đơn giản đối với thư viện đồ họa.
  - Người lập trình cần hiểu tham số để thiết lập cho đúng.
  - Các tham số đặc tả
    - Sáu bậc tự do (Six degrees of freedom)
      - Vị trí tâm của thấu kính
      - Hướng
    - Thấu kính
    - Kích thước phim
    - Hướng của mặt phẳng phim



# Lập trình đồ họa



- Đặc tả nguồn sáng và vật liệu (light & material)
  - Việc thiết lập khá đơn giản đối với thư viện đồ họa.
  - Người lập trình cần hiểu tham số để thiết lập cho đúng.
  - Các kiểu nguồn sáng
    - Nguồn sáng từ một điểm và nguồn sáng phân tán
    - Nơi chiếu sáng
    - Nguồn sáng xa và gần
    - Các thuộc tính màu
  - Các đặc tính vật liệu
    - Hấp thụ: các đặc tính màu
    - Phân tán
      - Khuyếch tán
      - Tạo bóng

# WebGL 2.0



- WebGL là một thư viện đồ họa triển khai trên ngôn ngữ Javascript.
- Được hỗ trợ bởi các trình duyệt web mới nhất.

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser	KaiOS Browser
		2-3.6	4-7	3.1-5	10-11.5											
	11 12-18	11 4-23	11 8-32	11 5.1-7.1	11 12.1-18	3.2-7.1										
6-10	79-84	24-79	33-84	8-13.1	19-69	8-13.7		2.1-4.4.4	12-12.1				4-11.2			
11	85	80	85	14	70	14.0	all	81	59	85	79	11 12.12	12.0	10.4	7.12	11 2.5
		81-82	86-88	TP												

- WebGL 2.0 yêu cầu phần cứng có hỗ trợ thư viện đồ họa OpenGL ES 3.0
  - OpenGL ES là phiên bản đơn giản hơn của OpenGL dành cho các hệ thống nhúng.
  - Từ phiên bản OpenGL ES 2.0 trở đi, thư viện đồ họa chạy trên **nền tảng Shader**

# GLSL



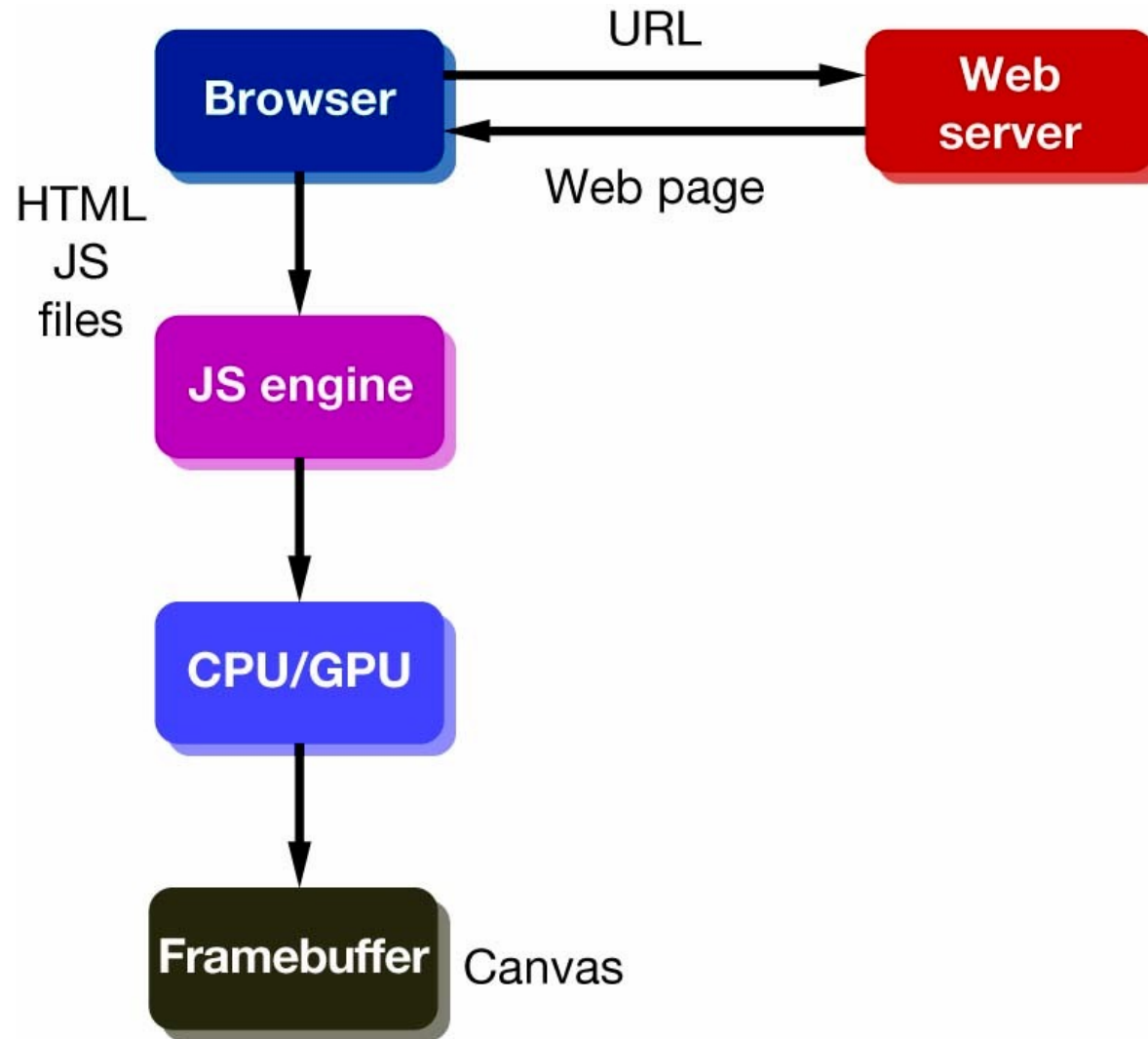
- GLSL (OpenGL Shader Language) là một ngôn ngữ tương tự ngôn ngữ C và được dùng để tạo ra các shader.

```
<script id="vertex-shader" type="x-shader/x-vertex">
  #version 300 es

  in vec2 vPosition;

  void main()
  {
    gl_Position = vec4(vPosition, 0.0, 1.0);
  }
</script>
```

# Thực thi chương trình trên WebBrowser



# Ví dụ vẽ hình tam giác



- Vẽ một hình tam giác màu đỏ
  - Mỗi chương trình gồm ít nhất 2 files
    - File HTML
      - Chứa mô tả của trang Web
      - Chứa các tiện ích, thư viện của chương trình
      - Chứa các shaders
    - File JavaScript
      - Chứa các đối tượng đồ họa



# Ví dụ vẽ hình tam giác



- Chạy file **index.html**
  - Hệ thống sẽ tự động nạp file **index.html** và **triangle.js** vào máy tính để chạy.
- Mở rộng: Sửa màu cho tam giác và chạy lại chương trình





# Các thư viện .JS



- `initShader.js`
  - Được dùng để đọc, biên dịch và liên kết các shader.
- `MV.js`
  - Được dùng để xử lý các vấn đề liên quan đến ma trận (Matrix) và véc tơ (Vector)



# triangle.js



- Cấu hình WebGL

```
gl.viewport(0, 0, canvas.width, canvas.height);  
gl.clearColor(1.0, 1.0, 1.0, 1.0);
```

- Nạp shader và khởi tạo các bộ đệm


```
var program = initShaders(gl, "vertex-shader", "fragment-shader");  
gl.useProgram(program);
```

- Đưa dữ liệu vào GPU

```
var vBuffer = gl.createBuffer();  
gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);  
gl.bufferData(gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW);
```

- Liên kết shader với buffer dữ liệu

```
var vPosition = gl.getAttribLocation(program, "vPosition");  
gl.vertexAttribPointer(vPosition, 2, gl.FLOAT, false, 0, 0);  
gl.enableVertexAttribArray(vPosition);
```



# Hết Tuần 1



Cảm ơn các bạn đã chú ý lắng nghe !!!