Đồ họa máy tính giải quyết tất cả những khía cạnh của việc tạo ra hình ảnh bằng máy tính

Là sự kết hợp của việc tạo ra hình ảnh bằng máy tính

Là sự kết hợp giữa các thành phần : Đối tượng đồ họa, phần mềm, phần cứng ( máy tính, card)

Ứng dụng: - Biểu diễn thông tin

* Hiển thị thông tin : Hình ảnh, linked data
* Hỗ trợ thiết kế
* Hệ thống giả lập
* VR – thực tế ảo, AR – thực thế tăng cường

Câu 1: **CẤU TẠO CỦA PIXEL? CÁCH TẠO RA PIXEL?**

* Pixel là đơn vị nhỏ nhất của ảnh, ảnh là 1 mảng 2 chiều có các điểm ảnh.
* Cấu tạo của pixel:
  + Vị trí của pixel: tọa độ X, Y.
  + Gía trị của pixel: màu của pixel.
* Cách tạo ra pixel:
  + Tính toán theo số bit, số lượng bit được sử dụng cho mỗi pixel sẽ xác định số lượng màu có thể được tạo ra cho ảnh
    - 1 bits: số lượng màu của ảnh là 2
    - 8 bits: số lượng màu của ảnh là 256
* Tạo lập hình ảnh cần – đối tượng, vật liệu, góc nhìn, nguồn sáng
* Pipeline : chia xử lý thành nhiều nhân độc lập, các bước xử lý đc thực hiện bên trong phần cứng, xử lý thông tin ở các đỉnh có thể xử lý song song
* Xử lý đỉnh là chuyển đổi mô tả đối tượng trên hệ trục tọa độ này sang hệ trục tọa độ khác – Tọa đội đối tượng, tọa độ camera, tọa độ màn hình
* Tất cả việc chuyển đổi tọa độ này tương đương việc biến đổi ma trận
* Xử lý đỉnh gồm cả việc tính toán màu sắc
* Dựa trên các đỉnh sẽ lên các hình: đoạn thẳng – đa giác – đường cong – bề mặt
* Các đối tượng không nằm trong vùng view volumn sẽ đc cắt xén
* Trong webgl ác đối tượng đc tạo thành bở nhiều mặt phẳng(fragment) với nhau
* Các màn đc xác định bởi texture hoặc nội suy theo màn đỉnh

Thư viện cung cấp các hàm tạo dựng và đặc tả: đối tượng(điểm, đg thẳng, đa giác), vật liệu, góc nhìn(phép chiếu), nguồn sáng, thiết bị đầu vào, các khả năng khác lquan đến hệ thống

Câu 2: **CÁC THÔNG TIN MÀU SẮC PIXEL HIỂN THỊ ĐƯỢC LƯU TRỮ Ở ĐÂU?**

* Được lưu trong Frame Buffer
* Frame Buffer: Input -> CPU(<-> RAM) -> GPU(<-> VRAM) -> Display( <- FRAME BUFFER)
* Frame Buffer nằm trong bộ nhớ máy tính hoặc bên trong GPU (graphic processing unit)
* Hình ảnh ở frame buffer là hình ảnh đc đưa lên màn hình
* Được sử dụng để lưu trữ điểm ảnh (pixel) trên máy tính
* Độ phân giải của Frame Buffer và độ phân giải của màn hình là giống nhau

Câu 3: **CÓ MẤY LOẠI SHADER? CHỨC NĂNG? NHIỆM VỤ?**

**Shader** là các chương trình nằm bên trong gpu nhằm xử lý 1 việc nào đó

* Có 2 loại Shader:
  + Vertex Shader:
    - Vertex shader là chương trình được gọi để biến đổi đỉnh. Nó được sử dụng để biến đổi hình học từ một nơi này sang nơi khác. Nó xử lý dữ liệu của mỗi đỉnh như: tọa độ, màu sắc, texture (cấu trúc bề mặt). Các nhiệm vụ của vertex shader bao gồm:
      * Biến đổi các đỉnh.
      * Áp dụng màu sắc.
      * Tạo ánh sáng.
      * Sinh ra texture
      * Biến đổi texture.
    - Chứa đoạn mã chương trình dùng để chạy trên mỗi đỉnh của đối tượng truyền vào
    - Vertex Shader xử lí dữ liệu của mỗi đỉnh như biến đổi tọa độ đỉnh, chuẩn hóa, xử lí màu
    - Sau khi nhận được dữ liệu truyền vào từ bên ngoài, điểm bắt đầu thực thi của chương trình là hàm Main().
  + Fragment Shader:
    - Một lưới đồ họa được ghép bởi nhiều hình tam giác, và bề mặt của mỗi một tam giác được hiểu là một mảnh (fragment). Fragment Shader là phần code chạy trên tất cả các điểm của một tất cả các mảnh. Nó thường dùng để tính toán và tạo màu cho các điểm. Các nhiệm vụ của Fragment Shader là:
      * Truy cập texture.
      * Áp dụng texture.
      * Tạo độ mờ.
      * Tính toán màu sắc
    - Chứa đoạn mã chương trình được thực hiện ngay sau khi hàm Main() bên trong Vertex Shader kết thức.
    - Có nhiệm vụ tính toán màu sắc của mỗi pixel liên quan đến đỉnh hoặc tổ hợp đỉnh.
    - Sau khi nhận được dữ liều truyền vào từ Vertex Shader , điểm bắt đầu thực thi của Fragment Shader là hàm Main()
    - Đầu ra hàm Main() của Fragment Shader là màu sắc đỉnh của Vertex Shader vừa xử lí.
    - Truy cập đến dữ liệu vectơ – qua chỉ số ( 0,1,2,3,) qua trục tọa độ ( x,y,z,w) qua tên tọa độ texture (s,t,p,q) thông qua tên màu sắc (r,g,b,a)

OpenGL Shading Language, là một quy chuẩn để viết các shader. GLSL k thể show lên các tính toán vì nó nằm trong đường ống đồ họa.

Câu 4: **VERTEX SHADER VÀ FREGMENT SHADER NHẬN DỮ LIỆU BẰNG CÁCH NÀO?**

Các dữ liệu được truyền cho vertex shader dạng buffer object được xử lý để render khi các phương thức sau được gọi drawElements() và drawArray(). Nhiệm vụ của vertex shader là tính toán vị trí của các vertex sau đó lưu vào biến gl\_position, ngoài ra nó cũng tính toán màu sắc, texture, và các đỉnh liên quan đến nhau.

Dữ liệu ta truyền vào các shader xử lý là các đỉnh của các hình học mà ta muốn vễ. Sau khi được các shader xử lý, quá trình primitive assembly sẽ diễn ra để nối các đỉnh lại tạo thành hình cần vẽ.

Vậy là công việc với vertices đầu vào đã hoàn tất. Việc tiếp theo là biểu diễn vị trí các vertices này trên màn hình. Đó là công việc của rastersizer.

Rastersizer(Đường ống kết xuất) sẽ nhận tọa độ vertices đầu vào (đã được biến đổi từ các bước trước) và generates fragment(s) cho các vertices đó.

Trong bước này, fragment shader sẽ lấy từng fragment từ rastersizer và tính toán các giá trị depth (chiều sâu) , stencil ( là 1 bộ đệm) và màu của fragment đó. Trong bước này, màu của một fragment sẽ được quyết định. Một điểm mạnh của fragment shader là chúng có thể được dùng với kĩ thuật texture mapping để áp texture lên bề mặt vật thể. Thay vì tô màu, chúng ta có thể "tô" texture.

* Vertex Shader nhận dũ liệu nhập vào thông qua gl\_Position
  + Gl\_Position:
    - Là biến được định nghĩa sẵn trong vertex shader, dùng để xác định vị trí đỉnh mà giá trị của nó xác định trên trục webGL
    - Sau khi được truyền giá trị, nó sẽ được sử dụng bởi các công việc tiếp theo trong đường ống đồ họa
* Rồi xử lí sau đó truyền dữ liệu đã xử lí cho Fragment Shader tiếp nhận rồi xử lí màu sắc rồi đẩy ra màn hình.

Câu 5: **CÓ NÊN DÙNG FRAGMENT SHADER THAY CHO VERTEXSHADER KHÔNG?**

* Không nên xử dụng Fragment Shader cho các thao tác biến đổi đỉnh vì bản chất Fragment Shader có nhiệm vụ xử lí đồ họa.

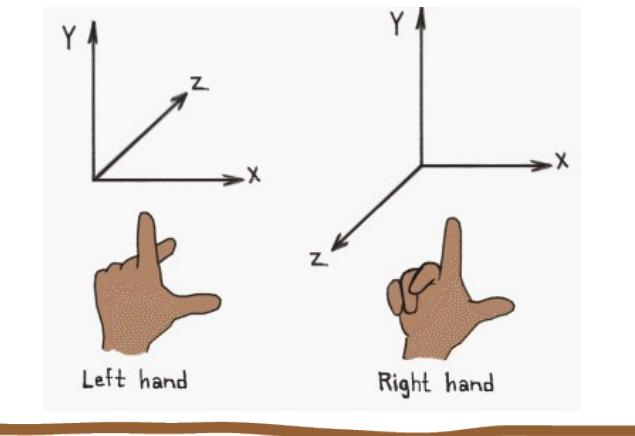
Câu 6: **NGUYÊN LÝ TẠO MÀU CHO MỘT ĐA GIÁC N ĐỈNH?**

* Đầu tiên tính toán chia đa giác n đỉnh thành các tam giác m đỉnh
* Vertex Shader:
  + Sẽ được thục hiện lặp lại m lần tương ứng với m đỉnh
  + Gl\_position sẽ chứ giá trịnh đính mới sau khi tính toán. GPU sẽ lưu lại giá trị trong gl\_Position để xứ lý tiếp
* Fragment Shader:
  + Để vẽ được 1 tam giác ta vừa chia ra thì ta phải dựa trên tọa độ màu và màu sắc của 3 pixel tương ứng với 3 đỉnh của tam giác.
  + Hệ thống sẽ tự động sinh ra fragment cho tam giác đó
  + Fragment Shader sẽ tô màu cho các frament này cụ thể là xác định màu cho từng pixel bên trong frament:
    - 3 đỉnh cùng màu: thì pixel bên trong tam giác sẽ trùng màu với đỉnh
    - 3 đỉnh khác màu: thì pixel bên trong tam giác sẽ được nội suy

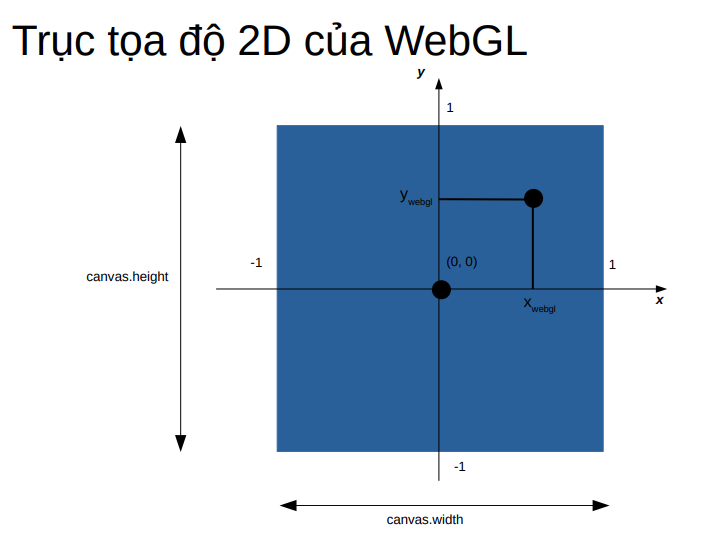
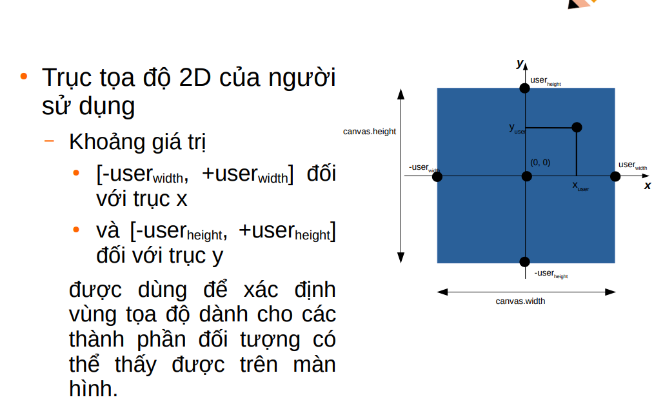
Câu 7: **CẤU TẠO CỦA TRỤC TỌA ĐỘ WEBGL TRÊN KHÔNG GIAN 3 CHIỀU?**

Hê tọa độ trong WebGL có 3 chiều x, y, z như hệ 3D, tuy nhiên trong đó z biểu diễn cho chiều sâu. Hệ tọa độ này bị giới hạn trong (1, 1, 1) đến (-1, -1, -1) nghĩa là nếu coi màn hình project WebGL là một hình lập phương thì một góc của nó có tọa độ là (1, 1, 1) còn góc đối diện qua tâm là (-1, -1, -1). WebGL sẽ không hiển thị bất cứ cái gì được vẽ ngoài giới hạn này.

* Cấu tạo trục WEBGL trên không gian 3 chiều gồm : X, Y , Z
  + Ngoài 2 trục là chiều rộng và chiều dài sẽ có thêm chiều sâu để thể hiện sự tương quan giữa các mặt của đối tượng
* Tuân theo quy tắc bàn tay trái:
* Người sử dụng là quy tắc bàn tay phải

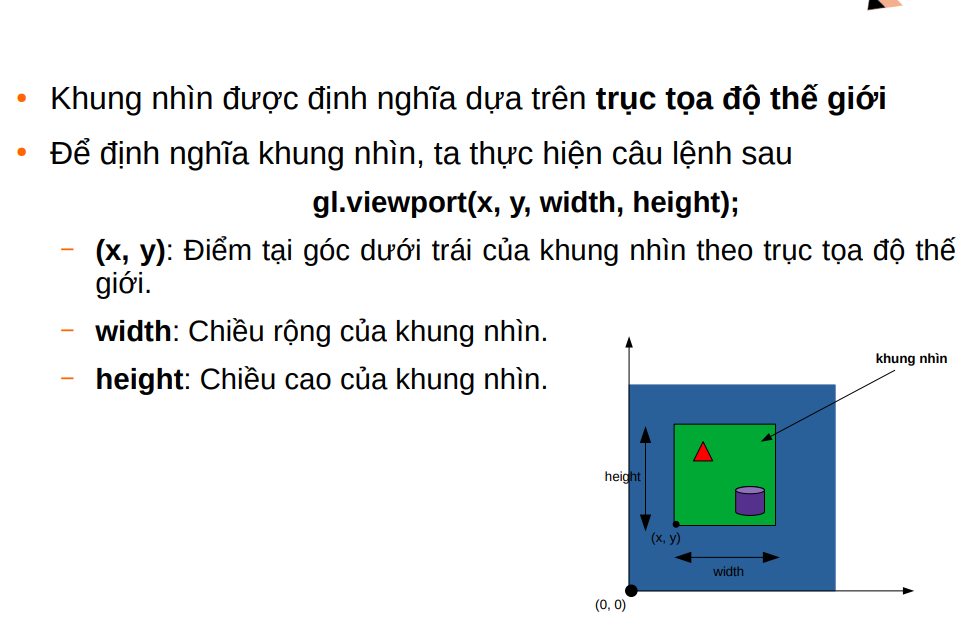


Câu 8: **BIẾN ĐỔI TRỤC TỌA ĐỘ NGƯỜI DÙNG SANG WEBGL?**

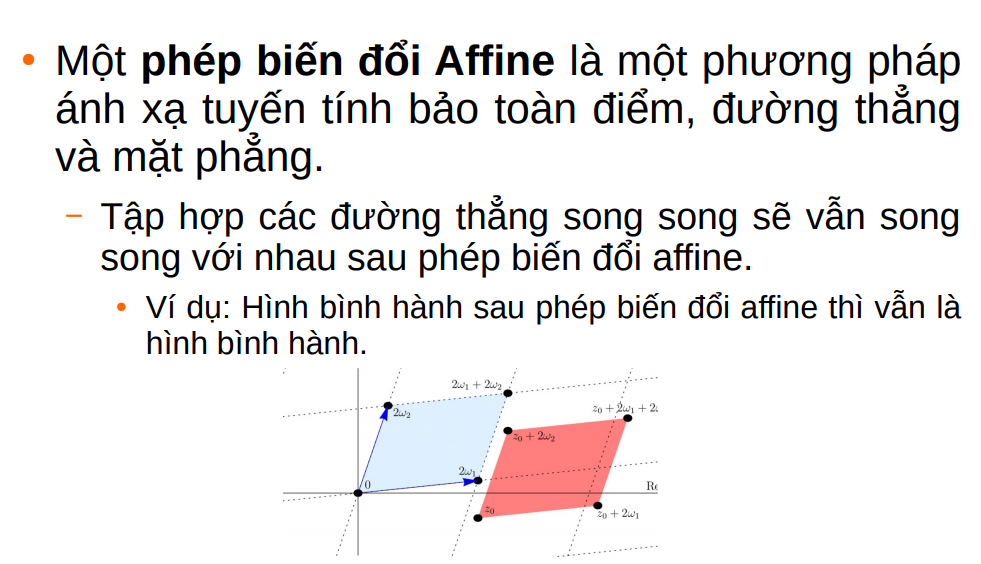


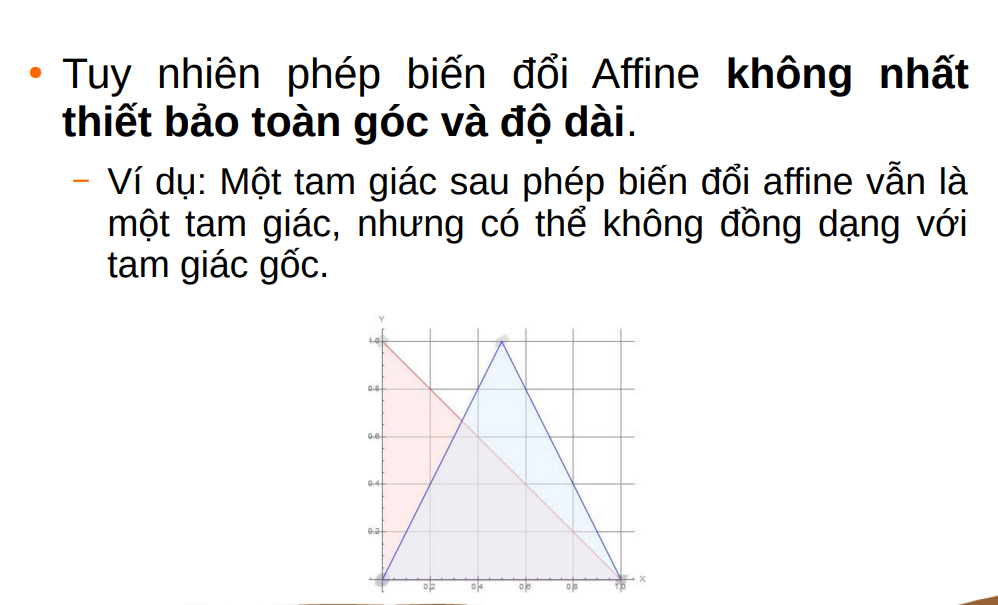
* Cách thức biến đổi từ tọa độ trục người dùng sang tọa độ WEBGL:
  + Trong WEBGL, tọa độ giới hạn từ -1 đến 1, muốn biến đổi tọa độ người dùng sang tọa độ của WebGL ta cần:
    - Biến đổi dựa trên công thức:
      * X = X(user) / User(width)
      * Y = Y(user) / User(height)
    - Trục tóa độ 2D người sử dụng
      * [-user(width), + user(width)] đối với trục x
      * [-user(height), + user(height)] đối với trục y
      * Được sử dụng để xác định vùng tọa độ dành cho các thành phần đối tượng có thể thấy trên màn hình

Câu 9: **HIỂU BIẾT VỀ KHUNG NHÌN (VIEWPORT)? ỨNG DỤNG?**

* Khung nhìn: được định nghĩa dựa trên trục tọa độ thế giới.
  + Là 1 khung cửa sổ gồm các đặc điểm sau:
    - Được định nghĩa bởi người dùng
    - Nằm bên trong Canvas
    - Dùng để chứa toàn bộ các đối tượng hình ảnh trên hệ trục tọa độ WebGL
  + Canvas có thể chứa nhiều khung hình nhưng chỉ WebGL chỉ thể hiện khung hình được định nghĩa sau cùng. 

Câu 10: **VÌ SAO NGƯỜI TA SỬ DỤNG PHÉP BIẾN ĐỔI AFFINE?**



* Phép biến đổi Affine:
  + Người ta dùng phép biến đổi Affine vì nó bảo toàn điểm, đường thẳng, mặt phẳng và tỷ lệ. Sau khi biến đổi vẫn giữ nguyên được bản chất tuy không thể giống 100% ban đầu.
  + Ví dụ: Đường thẳng song song vẫn sẽ song song sau khi qua phép biến đổi Affine.
  + 

Câu 11: **PHÉP BIẾN ĐỔI AFFINE CÓ BẢO TOÀN GÓC VÀ ĐỘ DÀI KHÔNG?**

* Tỷ lệ trong phép biến đổi này được bảo toàn
* Không nhất thiết bảo toàn góc và độ dài

Câu 12: **TRÌNH BÀY PHƯƠNG PHÁP XÂY DỤNG PHÉP BIẾN ĐỔI TỔ HỢP?**

* Phương pháp biến đổi hình tổ hợp:
  + Phép biến đổi tổ hợp dựa trên các phép biến đổi hình cơ bản như biến đổi tọa độ, biến đổi gốc, biến đổi hình dạng, …
  + B1: Phân tích phép biến đổi tổ hợp thành các phép biến đổi cơ bản
  + B2: Tìm ra ma trận biến đổi của các phép biến đổi cơ bản rồi lấy chúng nhân với nhau.

Câu 13: **PP XÁC ĐỊNH MẶT TRONG MẶT NGOÀI CỦA ĐA GIÁC TRÊN KHÔNG GIAN 3D?**

* Phương pháp xác định mặt trong, mặt ngoài trên không gian 3D:
  + Mặt trong, mặt ngoài của đa giác được thể hiện dựa trên thứ tự các đỉnh.
  + Sử dụng quy tắc bàn tay phải cho việc xác định mặt ngoài:
    - Để ngón cái của bàn tay phải vuông góc hướng về phía mặt mà bạn coi đó là mặt ngoài.
    - Khum và tạo độ cong cho 4 ngón tay còn lại, hướng của 4 ngón tay là hướng đi thứ tự của các đỉnh.

Câu 14: **PHÂN BIỆT CẤU TRÚC GIỮA TOPO VÀ ĐỈNH?**

* Cấu trúc hình học:
  + Vị trí của các đỉnh
* Cấu trúc topo:
  + Thể hiện mối liên hệ của các đỉnh và các cạnh.
* Khi cấu trúc hình học thay đổi, cấu trúc topo không thay đổi.

Câu 15: **VÌ SAO PHẢI XÂY DỤNG CẤU TRÚC 3D?**

* Ta nên xây dựng cấu trúc dữ liệu cho đối tượng 3D, vì ta có thể phân chia hình 1 cách mạch lạc và rõ ràng. Thuận tiện cho việc triển khai cho những hình lớn và có độ phức tạp cao.

Câu 16: **PHÉP CHIẾU SONG SONG VÀ PHÉP CHIẾU BỐI CẢNH?**

* Phép chiếu song song:
  + Được dùng để thu lại chính xác hình ảnh ở các khía cạnh khác nhau của đối tượng.
  + Tuy nhiên phép chiếu song song khi chiếu nên mặt phẳng 2 chiều sẽ không giống như thực tế
    - Độ lớn của hình chiếu không phụ thuộc vào khoảng cách chiếu.
  + Ứng dụng trong vẽ chi tiết lát cắt.
* Phép chiếu phối cảnh:
  + Được dùng để thu hình ảnh tụ về 1 điểm gọi là tâm chiếu.
  + Là phép chiếu có độ chân thực cao.
  + Các đường thẳng song song khi chiếu lên mặt phẳng thì chưa chắc đã song song.
  + Ứng dụng trong các bản vẽ tỉ lệ.

Câu 17: **PHÉP CHIẾU 1,2,3 ĐIỂM BIẾN MẤT?**

* Điểm biến mất là các đưởng thằng song song trên đối tượng thực hội tụ về 1 điểm trên hình chiếu.
* Một điểm biến mất:
  + Các đường thẳng ở đối tượng thực song song trên trục X và Y vẫn song song trên hình chiếu.
  + Các đường thẳng ở đối tượng thực song song trên trục Z sẽ hội tụ về 1 điểm trên hình chiếu.
* Hai điểm biến mất:
  + Các đường thẳng ở đối tượng thực song song trên trục Y vẫn giữ nguyên trên hình chiếu.
  + Các đường thẳng ở đối tượng thực song song trên trục X và Z sẽ hội tụ về 2 điểm tương ứng trên hình chiếu.
* Ba điểm biến mất:
  + Các đường thẳng ở đối tượng thực song song của cả 3 trục hội tụ về 3 điểm tương ứng trên hình chiếu.

Câu 18: **HIỂU BIẾT VỀ CÁCH TÍNH MA TRẬN MODEL-VIEW? MODEL VIEW ĐỂ LÀM GÌ?**

* Biến đổi Model-View là một ánh xạ dùng để chuyển đổi các giá trị từ trục tọa độ mắt người sử dụng sang trục tọa độ trong camera(mắt)
* Ma trận biến đổi Model:
  + Các đối tượng ban đầu được định nghĩa trong trục tọa độ người sử dụng
  + Định vị các đối tượng rồi biến đổi (xoay, tịnh tiến, tỷ lệ, …) đều được thực hiện bên trong trục tọa độ người sử dụng.
* Ma trận biến đổi View:
  + Sau khi thiết lập camera trên trục tọa độ người sử dụng, ta cần chuyển đổi tọa độ của đối tượng sử dụng sang trục tọa độ camera (gốc của trục tọa độ camera chính là điểm nhìn)
* Biến đổi Model-View:
  + M là thao tác biến đổi Model
  + V là thao tác biến đổi Vierw
  + Ma trận MV (là tích 2 ma trận V và M) là ma trận được sử dụng để mỗi khi đối tượng ở trục tọa độ người sử dụng ta sẽ nhận được ngay vị trí mới của đối tượng ở trục tọa độ camera.

Câu 19: **TRONG ĐƯỜNG ỐNG ĐỒ HỌA BIẾN ĐỔI MODELVIEW ĐƯỢC XỬ LÝ TRƯỚC HAY SAU PHÉP CHIỀU?**

* Biến đổi phép chiếu được thực hiện sau khi chuyển đổi giá trị các đỉnh của đối tượng từ trục tọa độ của người sử dụng sang trục tọa độ camera (biến đổi ModelView)

Câu 20: **CHUẨN HÓA THỂ TÍCH NHÌN LÀ GÌ? THỂ TÍCH NHÌN CHUẨN LÀ GÌ?**

* Thể tích nhìn được chuẩn hóa (normalized view volume): là một hình lập phương có trọng tâm trùng với gốc tọa độ và cạnh có chiều bằng 2
* Chuẩn hóa thể tích: là biến đổi tất cả các thể tích nhìn về thể tích nhìn chuẩn
* Chuẩn hóa phép chiếu là biến đổi tất cả các phép chiếu về phép chiếu trục giao

Câu 21: **TRÌNH BÀY VỀ HÀM LOOKAT? Ý NGHĨA?**

* Hàm lookAt trong thư viện Mv.js được sử dụng để đặt điểm nhìn, hướng nhìn và góc xoay của camera.
* Cú pháp: lookAt(vec3 eye, vec3 at, vec3 up)
  + eye: Vị trí đặt điểm nhìn
  + at: at là tọa độ của một điểm mà véctơ có gốc là điểm nhìn eye và đỉnh là tọa độ at xác định hướng nhìn.
  + up: xác định góc xoay của camera

hàm này trả về 1 ma trận model view ( nối eye với at sẽ tạo 1 vecto xác định trục tọa độ của camera)

