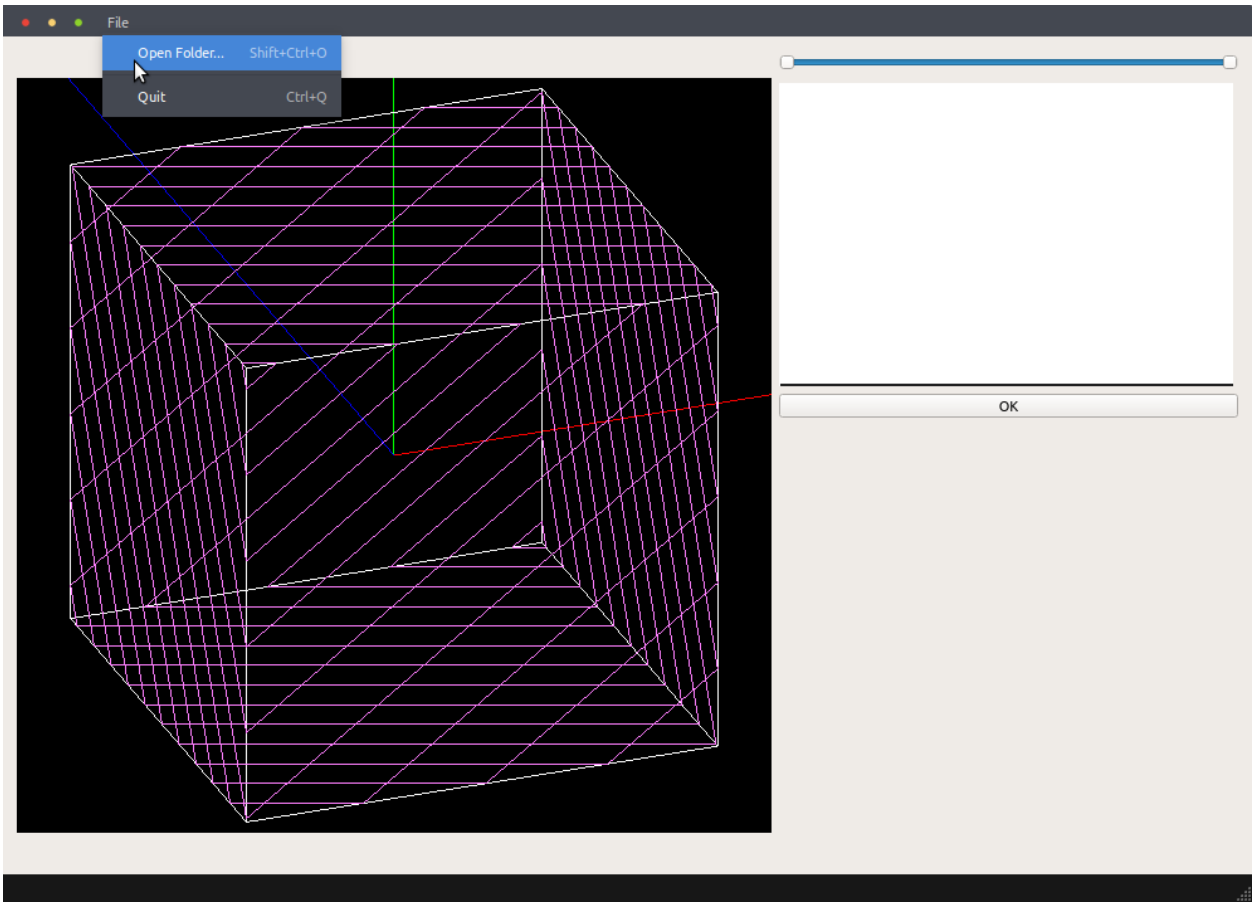
# ASSIGNMENT 2 – DICOM VISUALIZATION

Nguyễn Đức Thọ
Đồ hoạ máy tính
Đại học Bách Khoa TP. Hồ Chí Minh

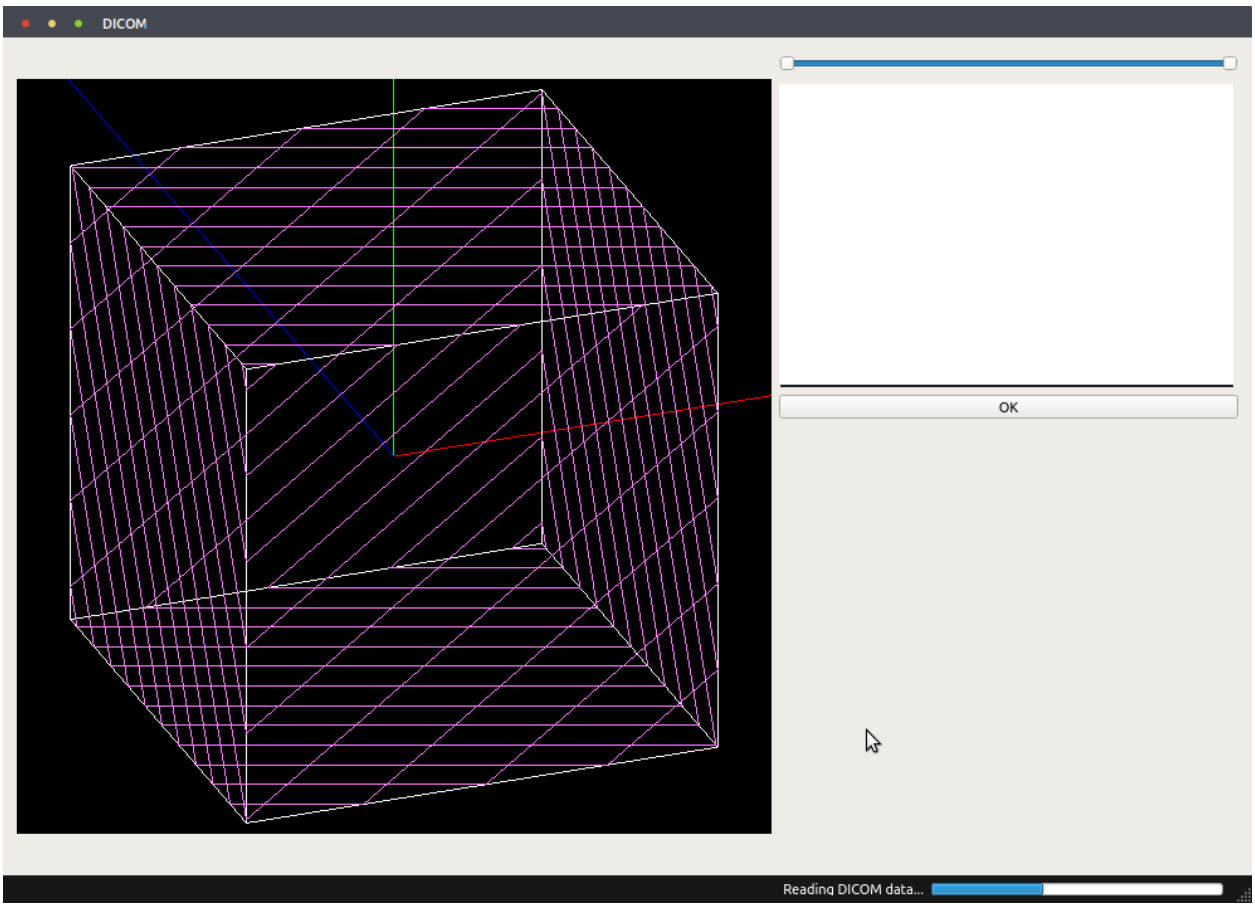
# Mục lục

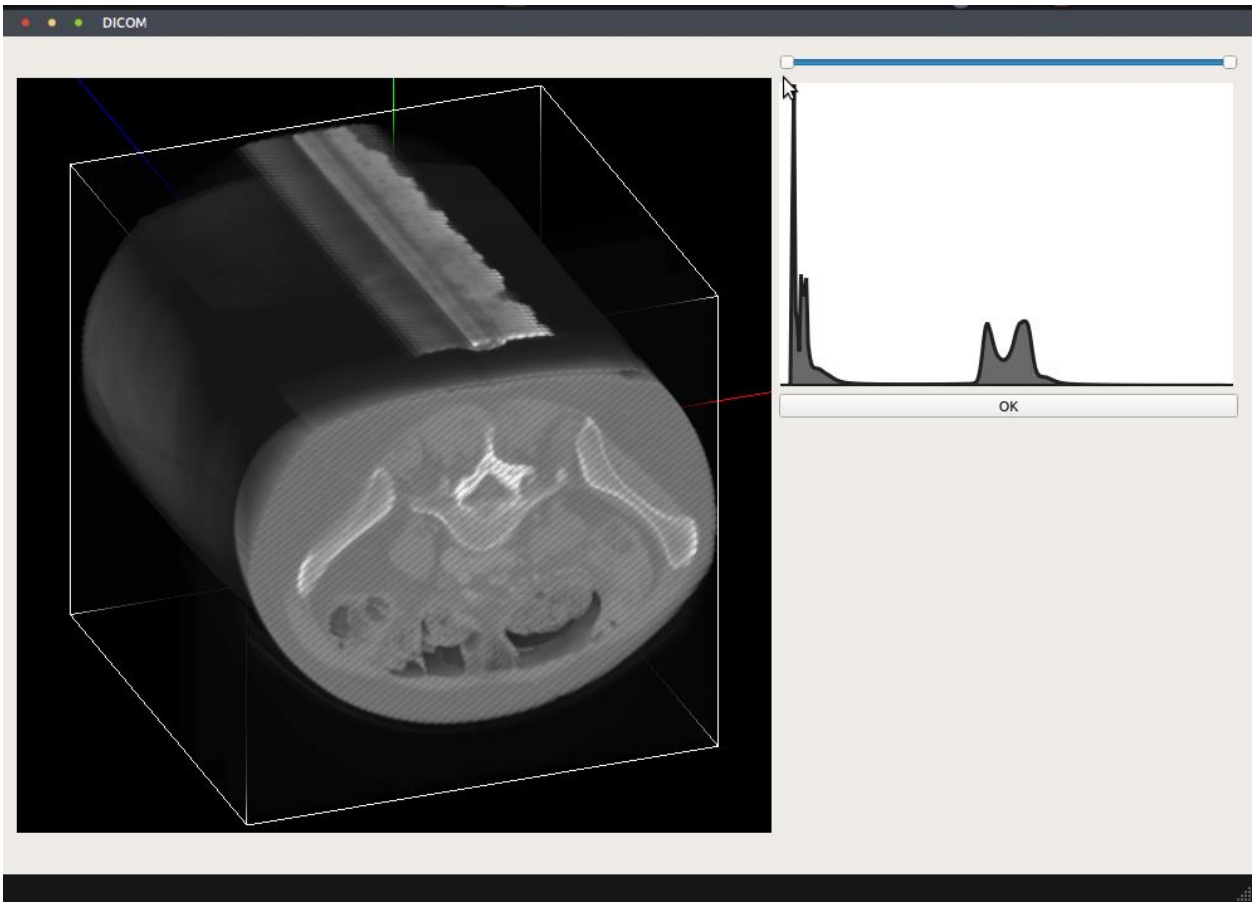
## 1. Giới thiệu chương trình



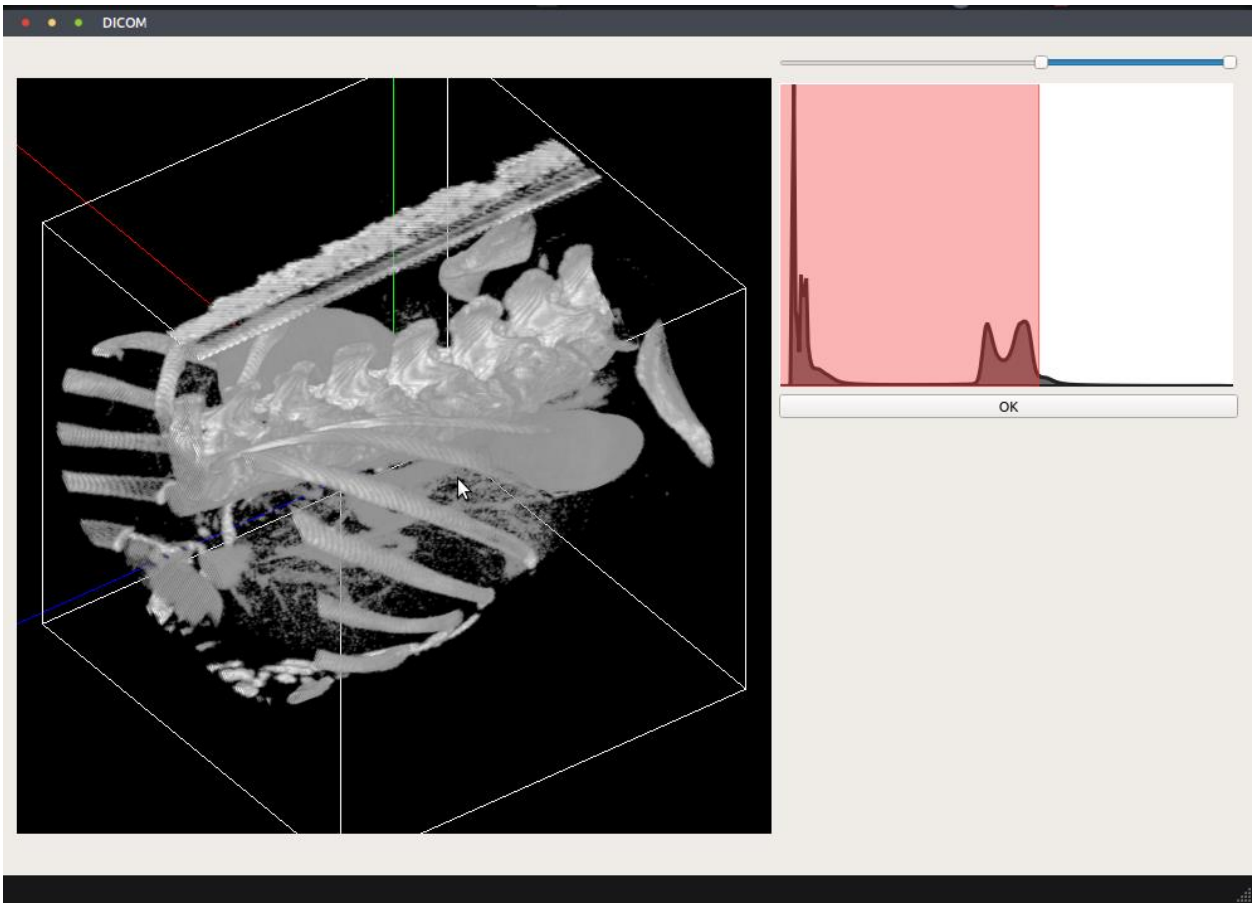

Khi khởi chạy, chương trình mặc định vẽ 10 đa giác, lần lượt nằm trong các mặt phẳng song song với nhau, và cùng vuông góc với tia nhìn. Tia nhìn này đi qua 2 điểm, vị trí camera và điểm nhìn (0, 0, 0). Khi xoay khối lập phương, các đa giác được tính và vẽ lại.

*File -> Open Folder…* để mở thư mục chứa các file DICOM, chương trình đọc và hiển thị.


DICOM
OK
Reading DICOM data...

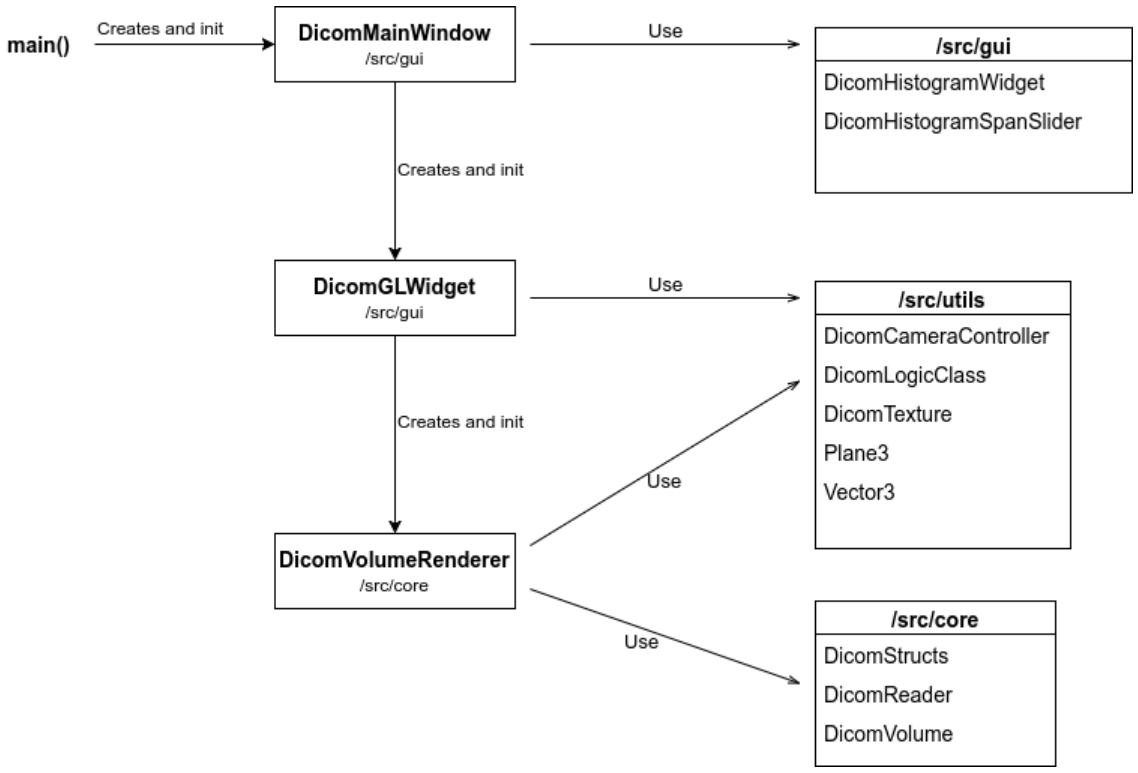

DICOM
OK

Chọn vùng histogram (màu trắng) và nhấn *OK* để xem.




## 2. Chi tiết hiện thực

### a. Kiến trúc chương trình



## b. Chức năng các class

- **DicomStructs**: khai báo cấu trúc dữ liệu.

```
typedef struct
{
   std::vector<DicomSingle *>    data;
   std::uint32_t                 width;
   std::uint32_t                 height;
   std::uint32_t                 depth;
   std::uint64_t *               histogram;
} DicomSet;
```

```
typedef struct
{
   std::vector<int>              luminance;
   std::vector<std::uint8_t>     luminance8;
   std::uint32_t                 width;
   std::uint32_t                 height;
   std::uint32_t                 instanceNumber;
   int                           smallestPixelValue;
   int                           largestPixelValue;
} DicomSingle;
```

- **DicomVolumeRenderer**: class chính thực hiện việc render hình ảnh DICOM. Đầu tiên, lấy vị trí camera và điểm nhìn từ DicomCameraController, tạo các mặt phẳng vuông góc với tia nhìn, sử dụng hàm *intersectPlane* trong DicomVolume để lấy danh sách giao điểm, sau đó gán texcoords vào các điểm đó.
- **DicomReader**: hỗ trợ đọc dữ liệu từ file DICOM, có sử dụng thêm thư viện *imebra*.
- **DicomVolume**: class biểu diễn khối lập phương mà chương trình sẽ render ảnh DICOM ra.

  std::vector <Point3d> intersectPlane(const Plane3d &plane) : trả về danh sách giao điểm giữa các cạnh của khối lập phương và một mặt phẳng bất kì. Danh sách trả về được sắp xếp ngược chiều kim đồng hồ.
- **DicomCameraController**: điều khiển việc xoay camera khi di chuyển chuột, trả về vị trí hiện tại của camera.
- **DicomTexture**: bind 3D texture từ dữ liệu đã được xử lí từ *DicomSet*.

**c. Xử lí dữ liệu đọc từ file DICOM**

```
                Filename: '/home/ndtho8205/Desktop/DICOM/data/3D…'
             FileModDate: '03-Th01-2017 02:38:00'
                FileSize: 171730
                  Format: 'DICOM'
           FormatVersion: 3
                   Width: 292
                  Height: 292
                BitDepth: 16
               ColorType: 'grayscale'

       AcquisitionNumber: []
          InstanceNumber: 100
    ImagePositionPatient: [3×1 double]
 ImageOrientationPatient: [6×1 double]
     FrameOfReferenceUID: '1.3.6.1.4.1.9590.100.1.2.38823913102…'
PositionReferenceIndicator: ''
         SamplesPerPixel: 1
PhotometricInterpretation: 'MONOCHROME2'
                    Rows: 292
                 Columns: 292
           PixelSpacing: [2×1 double]
          BitsAllocated: 16
             BitsStored: 16
                HighBit: 15
      PixelRepresentation: 1
 SmallestImagePixelValue: -1088
  LargestImagePixelValue: 1124
```

Dữ liệu mẫu của chương trình là một tập các file DICOM tạo thành một đối tượng 3D, trong đó, mỗi file là một slice 2D. Ngoài lưu thông tin về bệnh nhân, thiết bị được sử dụng,... file DICOM còn chứa một mảng 2D lưu các giá trị pixel.

Ảnh DICOM mẫu sử dụng 16-bit màu với color space là MONOCHROME2, trong khi thông thường màn hình máy tính chỉ chấp nhận 8-bit màu RGB (hoặc RGBA). Vì vậy cần xử lí các giá trị pixel trước khi render.

$$pixel_{8bit} = \frac{(pixel_{16bit} - SmallestImagePixelValue) * 255}{LargestImagePixelValue - SmallestImagePixelValue}$$

Công thức trên scale pixel từ 16-bit màu về 8-bit grayscale (giá trị từ 0 đến 255), và từ color space Grayscale chuyển sang RGBA bằng cách cho các giá trị R, G, B, A bằng $pixel_{8bit}$ (xem thêm tài liệu tham khảo 4).

$pixel_{16bit}$ được lưu trong trường *luminance*, $pixel_{8bit}$ được lưu trong trường *luminance8*.

Trường *instanceNumber* dùng để sắp xếp các *DicomSingle* theo thứ tự tăng dần đúng với thứ tự các lát cắt khi quét CT.

## 3. Render volume sử dụng 3D textures

Với 3D texture, khối lập phương được chia thành nhiều slice. Các slice này được tính bằng cách lấy giao điểm giữa các cạnh của khối lập phương với tập các mặt phẳng song song với mặt phẳng camera. Các slices được tính lại mỗi khi vị trí camera thay đổi.

### a. Load dữ liệu DICOM vào 3D texture

Được hiện thực trong method *load* của class *DicomTexture*.

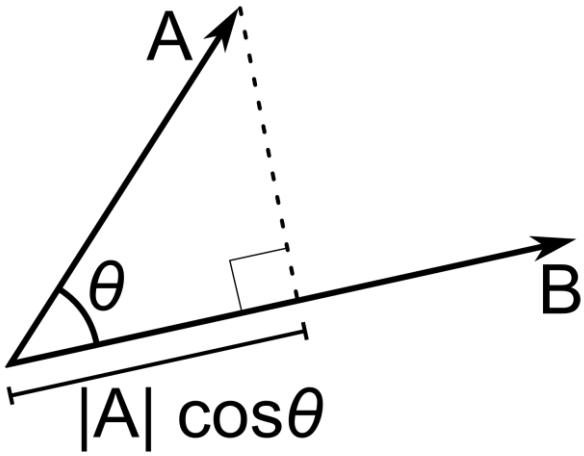
### b. Tìm đoạn thẳng giới hạn để vẽ tập các mặt phẳng song song:

Được hiện thực trong method *renderTextureDynamic* của class *DicomVolumeRenderer*.

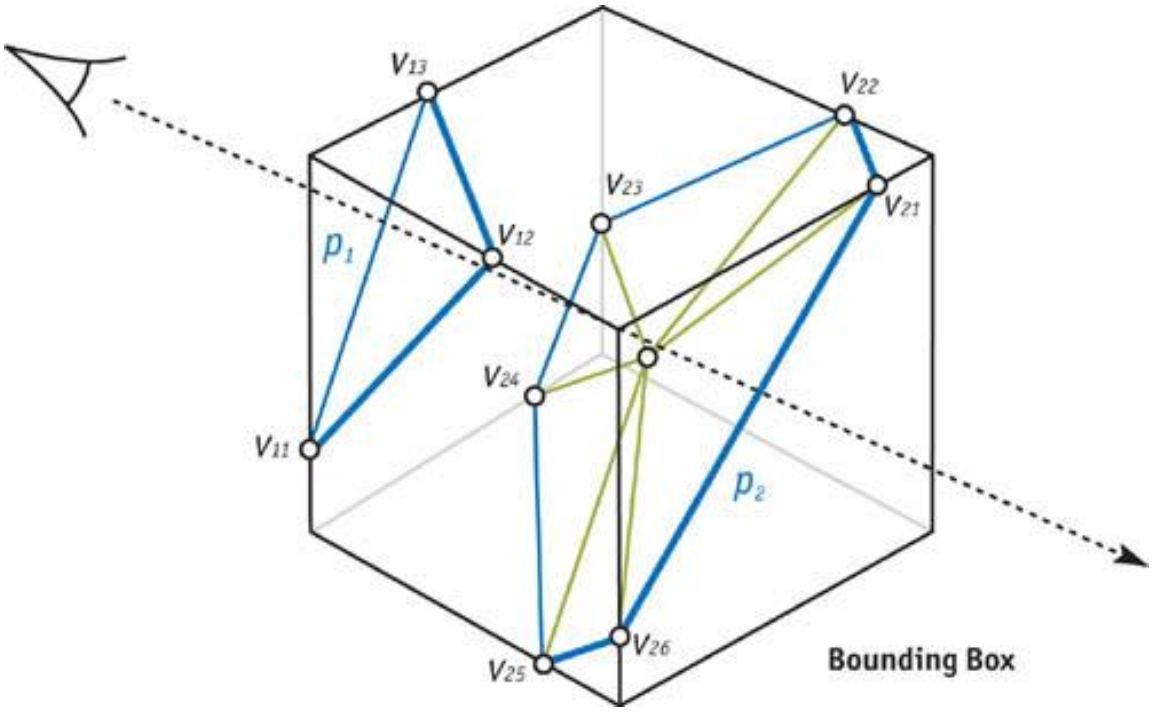
Tìm hình chiếu của các đỉnh của khối lập phương trên tia nhìn, lấy 2 điểm gần nhất và xa nhất. Ta được đoạn thẳng giới hạn việc vẽ các mặt phẳng song song với mặt phẳng camera, đảm bảo được việc các mặt phẳng này đều cắt các cạnh của khối lập phương.

Công thức tính toạ độ P là hình chiếu của A trên AB:

$$P = A + \frac{\overrightarrow{AP} \cdot \overrightarrow{AB}}{\overrightarrow{AB} \cdot \overrightarrow{AB}} * \overrightarrow{AB}$$




### c. Tính chuỗi các đa giác cắt ngang khối lập phương



Được hiện thực trong method *intersectPlane* của class *DicomVolume*.

Các giao điểm giữa khối lập phương và một mặt phẳng tạo thành một đa giác có từ 3 đến 6 đỉnh (trong trường hợp các đỉnh thuộc các cạnh của khối lập phương). Các giao điểm được tính bằng cách cho lần lượt các cạnh của khối cắt mặt phẳng.

Cạnh là một tập con của đường thẳng. Khi viết dưới dạng phương trình tham số, tìm giao điểm giữa cạnh AB và mặt phẳng ta được giá trị 0 <= *t* <= 1. Toạ độ của giao điểm đó là A + *t**B.

Sau khi tìm được tập giao điểm, ta sắp xếp lại theo thứ tự ngược chiều kim đồng hồ nhằm tăng tốc việc render, chỉ vẽ front của polygon.

**d. Gán texcoords**

Được hiện thực trong các methods *renderTextureDynamic*, *setTexture*, *setTextureCoordinates* của class *DicomVolumeRenderer*.

Lấy một điểm nằm giữa polygon (tính trung bình các toạ độ) rồi sử dụng GL_TRIANGLE_FAN gán texcoord cho các sđỉnh.

**e. Rendering**

Tắt GL_DEPTH_TEST, và enable GL_BLEND để trộn màu dựa theo giá trị alpha A.

Khi vị trí camera thay đổi, các slices được tính và vẽ lại.

## 4. Tài liệu tham khảo


1. 2007. “GPU Gems”. In *Volume Rendering Techniques*, chapter 39.
2. Christof Rezk Salama and Andreas Kolb, 2005. “A Vertex Program for Efficient Box-Plane Intersection”.
3. ftp://ftp.sgi.com/opengl/contrib/blythe/advanced99/notes/node299.html
4. Công thức chuyển từ 16-bit sang 8-bit grayscale được tham khảo từ file Conv16to8.cpp trong 16to8.zip, Andrey_Dmitriev. Link