**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**LỚP CỬ NHÂN TÀI NĂNG**

**VŨ TUẤN HƯNG – NGUYỄN VINH TIỆP – HUỲNH QUỐC TRÍ**

**NGHIÊN CỨU VÀ PHÁT TRIỂN THỬ NGHIỆM MỘT SỐ PHƯƠNG PHÁP TƯƠNG TÁC VỚI MÁY TÍNH SỬ DỤNG THỊ GIÁC MÁY TÍNH**

**KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT**

**TP.HCM, 20****10**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**LỚP CỬ NHÂN TÀI NĂNG**

**VŨ TUẤN HƯNG 0612151**

**NGUYỄN VINH TIỆP 0612450**

**HUỲNH QUỐC TRÍ 0612483**

**NGHIÊN CỨU VÀ PHÁT TRIỂN THỬ NGHIỆM MỘT SỐ PHƯƠNG PHÁP TƯƠNG TÁC VỚI MÁY TÍNH SỬ DỤNG THỊ GIÁC MÁY TÍNH**

**KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN TIN HỌC**

**GIÁO VIÊN HƯỚNG DẪN**

**TS.Trần Minh Triết**

**NIÊN KHÓA 2006 – 2010**

|  |
| --- |
| **NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  Khóa luận đáp ứng yêu cầu của LV cử nhân tin học.  TpHCM, ngày …… tháng …… năm 2010  Giáo viên hướng dẫn |
| **NHẬN XÉT CỦA GIÁO VIÊN PHẢN BIỆN**  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  ………………………………………………………………………………  Khóa luận đáp ứng yêu cầu của LV cử nhân tin học.  TpHCM, ngày …… tháng …… năm 2010  Giáo viên phản biện |

**LỜI CÁM ƠN**

Chúng em xin chân thành cảm ơn Khoa Công Nghệ Thông Tin, trường Đại Học Khoa Học Tự Nhiên, Tp.HCM đã tạo điều kiện tốt cho chúng em thực hiện đề tài này.

Chúng em xin chân thành cảm ơn Thầy Trần Minh Triết là người đã tận tình hướng dẫn, chỉ bảo chúng em trong suốt thời gian thực hiện đề tài.

Chúng em cũng xin gửi lời cảm ơn sâu sắc đến quý Thầy Cô trong Khoa đã tận tình giảng dạy, trang bị cho chúng em những kiến thức quí báu trong những năm học vừa qua.

Chúng em xin gửi lòng biết ơn sâu sắc đến Ba, Mẹ, các anh chị và bạn bè đã ủng hộ, giúp đỡ và động viên chúng em trong những lúc khó khăn cũng như trong suốt thời gian học tập và nghiên cứu.

Xin chân thành cám ơn Thầy Nguyễn Khắc Huy, bạn Nguyễn Đức Hoàng, các em khóa 2008 trong nhóm Smart Digital Content (Lê Hoàng Ân, Phạm Trường An, Mạc Cự Khôi Nguyên, Vũ Đức Quang Minh, Nguyễn Hưng, Đoàn Minh Thông) đã hỗ trợ cho nhóm trong quá trình thực hiện và thử nghiệm đề tài.

Mặc dù chúng em đã cố gắng hoàn thành luận văn trong phạm vi và khả năng cho phép, nhưng chắc chắn sẽ không tránh khỏi những thiếu sót, kính mong sự cảm thông và tận tình chỉ bảo của quý Thầy Cô và các bạn.

Nhóm thực hiện

Vũ Tuấn Hưng – Nguyễn Vinh Tiệp & Huỳnh Quốc Trí

**ĐỀ CƯƠNG CHI TIẾT**

|  |  |
| --- | --- |
| **Tên Đề Tài**: Nghiên cứu và phát triển thử nghiệm một số phương pháp tương tác với máy tính sử dụng thị giác máy tính | |
| **Giáo viên hướng dẫn:** TS.Trần Minh Triết | |
| **Thời gian thực hiện**: từ ngày 15/12/2009 đến ngày 15/07/2010 | |
| **Sinh viên thực hiện:**  0612450 - Nguyễn Vinh Tiệp  0612151 - Vũ Tuấn Hưng  0612483 - Huỳnh Quốc Trí | |
| **Loại đề tài**: Nghiên cứu lý thuyết về thị giác máy tính trong lĩnh vực tương tác người - máy, tìm hiểu công nghệ và xây dựng ứng dụng thử nghiệm | |
| **Nội Dung Đề Tài:** Khảo sát, nghiên cứu, phân tích một số phương pháp tương tác người – máy sử dụng thị giác máy tính; đề xuất giải pháp tương tác máy tính để sử dụng một và hai camera; từ đó xây dựng thử nghiệm một số ứng dụng cho phép tương tác giữa người – máy.  Nội dung chi tiết của đề tài bao gồm:   * Nghiên cứu, khảo sát các kỹ thuật HCI, kiến trúc hệ thống HCI * Một số vấn đề về camera: mô hình và các xác định các tham số của camera * Hai bài toán quan tâm: Dựng đối tượng 3D ảo dựa trên đối tượng thật sử dụng một camera, tương tác máy tính dựa vào thông tin 3D của đối tượng sử dụng 2 camera.   + ***Thực tại ảo tăng cường sử dụng một camera để dựng một thành phố 3D ảo***: mục tiêu là nhận diện và tái tạo thông tin 3D của đối tượng (các lá bài) và đặt trên đó những kiến trúc ứng với từng loại lá bài. Người dùng có thể tùy ý thay đổi sắp xếp lại vị trí qua lại lên xuống các kiến trúc. Ý nghĩa thực tế của ứng dụng là xây dựng một thành phố 3D ảo với sự tương tác rất đơn giản từ phía người dùng, tạo sự tiện lợi cho việc thiết kế và triển khai những dự án xây dựng, các công trình kiến trúc.   + ***Hệ thống tương tác với máy tính bằng chuột ảo, sử dụng thông 3D tái tạo từ bàn tay của 2 camera***: mục tiêu của ứng dụng là giả lập việc sử dụng chuột bằng các chuyển động tương ứng của bàn tay và ngón tay. Hệ thống sẽ chỉ sử dụng 2 camera với chi phí thấp để thực hiện rút trích thông tin của tay. Ý nghĩa thực tế của ứng dụng này là tìm một cách thay thế việc sử dụng chuột thông thường bằng một phương thức thuận tiện hơn với người dùng. * Ứng dụng thử nghiệm:   + Trò chơi “Lá bài ma thuật” dựa trên trò chơi “Eye of Judgement” của hãng Sony   + Ứng dụng chuột ảo từ thông tin 3D rút trích từ 2 camera,   + Ứng dụng Surface tương tác trực tiếp với màn hình máy tính sử dụng 2 camera | |
| **Kế Hoạch Thực Hiện**:   * 15/12/2009- 14/01/2010: Khảo sát về các hướng tiếp cận trong lĩnh vực tương tác người – máy, tập trung vào các nghiên cứu sử dụng Thị giác máy tính. * 15/01/2009- 14/02/2010: Tìm hiểu mô hình camera và cách xác định các tham số (trong và ngoài) của camera * 15/02/2009- 14/03/2010: Nghiên cứu và xây dựng giải pháp để lấy thông tin vị trí trong không gian 3 chiều của đối tượng đã được đăng ký trước trong hệ thống từ 1 camera * 15/03/2009- 14/04/2010: Nghiên cứu và xây dựng giải pháp để xác định vị trí trong không gian 3 chiều của bàn tay và các hành động của bàn tay với hình ảnh ghi nhận từ 2 camera * 15/04/2009- 14/05/2010: Phát triển thử nghiệm ứng dụng minh họa “Lá bài ma thuật” và phát triển thử nghiệm hệ thống framework hỗ trợ tương tác với máy tính sử dụng thông tin ghi nhận từ 2 camera * 15/05/2009- 14/06/2010: Phát triển thử nghiệm ứng dụng chuột ảo và surface * 15/06/2009- 14/07/2010: Hoàn thiện luận văn | |
| **Xác nhận của GVHD** | **Ngày 15 tháng 7 năm 2010 SV Thực hiện** |

**Mục lục**

🙢🕮🙠

[Chương 1 Giới thiệu 1](#_Toc297369920)

[1.1 Đặt vấn đề 1](#_Toc297369921)

[1.2 Mục tiêu của đề tài 4](#_Toc297369922)

[1.3 Nội dung luận văn 5](#_Toc297369923)

[Chương 2 Cấu hình kineck để thu nhận dữ liệu và xử lý nhiễu 6](#_Toc297369924)

[2.1 Kĩ thuật lấy thông tin ảnh màu và ảnh độ sâu từ kineck 6](#_Toc297369925)

[2.2 khử nhiễu ảnh màu 6](#_Toc297369926)

[Tài liệu tham khảo 9](#_Toc297369927)

**Danh sách hình**

🙢🕮🙠

[Hình 1.1 Những thành tựu trong HCI từ những năm đầu Error! Reference source not found.] 2](#_Toc297448721)

[Hình 2.1 mặt nạ màu thông dụng của bộ lọc Bayer[ 2][ 4] 9](#_Toc297448722)

[Hình 2.2 Ảnh màu (được phóng to 4 lần) trước và sau khi sử dụng bộ lọc Bayer (thuật toán nội suy EdgeAware) 10](#_Toc297448723)

[Hình 3.1Minh họa 3 frame trên cùng 1 hệ trục của kineck. 12](#_Toc297448724)

[Hình 3.2 Minh họa việc dùng Ransac áp dụng cho việc tìm đường thẳng trong mặt phẳng[ 8]. 15](#_Toc297448725)

[Hình 3.3 Minh họa kết quả ghép 2 đám mây sau khi áp dụng ransac và Levenberg–Marquardt để xác định ma trận biến đổi. 16](#_Toc297448726)

Danh sách bảng

🙢🕮🙠

**No table of figures entries found.**

# Giới thiệu

**Tóm tắt chương:**

*✍ Nội dung chương 1 trình bày tổng quan về luận án, mục tiêu của luận án.* *Nội dung tóm tắt của từng chương trong luận án được trình bày ở cuối phần này.*

## Đặt vấn đề

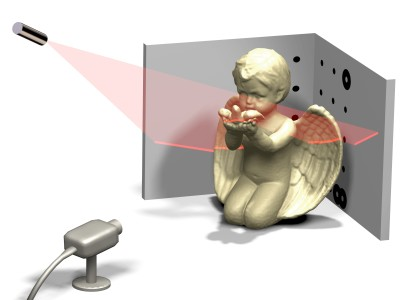
## Mục tiêu của đề tài

## Nội dung luận văn

# Mở đầu

## Giới thiệu chung

## 3D Scanner

3D Scanner là một loại thiết bị dùng để thu thập thông tin về hình dạng, màu sắc của các vật thể và môi trường xung quanh trong thế giới thực [ 6]. Thông thường, thông tin do 3D Scanner thu được là một đám mây điểm tương ứng với bề mặt của vật thể. Các thông tin này sẽ được sử dụng để tạo ra một mô hình 3D có hình dạng gần giống với vật thể trong thế giới thực. Hình 1‑1

Hình . ZScanner 700

Hình . Scan đối tượng

<http://www.hardwaresphere.com/2009/09/18/zscanner-700-3d-scanner/>

<http://gfxne.ws/2011/02/david-laserscanner-3d-laser-scanning/>

Tùy thuộc vào công nghệ được sử dụng mà 3D Scanner được chia thành hai loại chính : là Contact và Non-Contact.

### Contact 3D Scanner

Contact 3D Scanner lấy thông tin về hình dạng vật thể thông qua tiếp xúc về mặt vật lý nên rất chính xác. Vì thế, người ta thường sử dụng 3D Scanner loại này khi mô hình hóa các chi tiết phức tạp, đòi hỏi có độ chính xác cao. Tuy nhiên, 3D Scanner loại này cũng có nhiều nhược điểm như chậm và có thể làm hư hỏng vật thể do va chạm.

### Non-Contact 3D Scanner

Non-Contact 3D Scanner được chia thành hai nhóm chính : nhóm chủ động (active) và nhóm thụ động (passive).

* Scanner thuộc nhóm chủ động sẽ phát ra các tia bức xạ hoặc ánh sáng nhìn thấy đến vật thể hoặc môi trường xung quanh và thu lại các tia phản hồi để từ đó tính được khoảng cách đến các bề mặt xung quanh. Các bức xạ được sử dụng có thể là tia x, tia tử ngoại hoặc sóng siêu âm ...
* Scanner thuộc nhóm thụ động không phát ra gì cả mà chi thu các tia xuất phát từ vật thể. Các tia này có thể là ánh sáng nhìn thấy hoặc tia hồng ngoại. Hầu hết các Scanner thuộc nhóm thụ động chỉ là các camera thông thường. Người ta dùng các camera này để chụp đối tượng ở nhiều góc độ khác nhau hoặc ở những điều kiện khác nhau để ước lượng được vị trí và hình dạng của vật thể so với camera.

## Sự ra đời và phát triển của của kinect

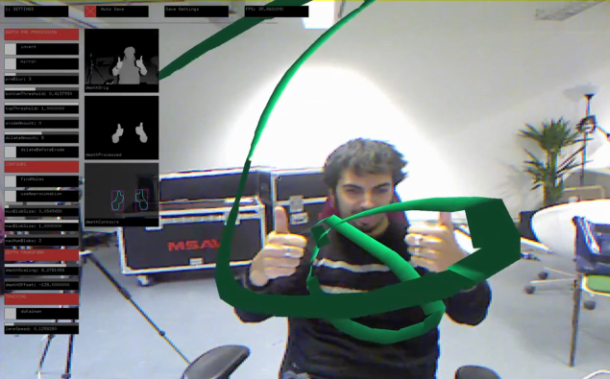


Hình . Giới thiệu Project Natal ở hội nghị E3 (2009)

Kinect (hay còn biết với mã là Project Natal) là 1 thiết bị thu nhận các cảm biến chuyển động, được phát triển bởi Microsoft [ 1]. Thời gian đầu, Kinect là thiết bị chuyên dụng của hệ máy Xbox 360, phục vụ cho việc chơi game.

Các mốc thời gian ra đời, phát triển của kinect và các thành phần liên quan:

* **5/30/2007:** Microsoft nung nấu ý tưởng về 1 thiết bị dùng camera ghi nhận cử động điều khiền thay cho các thiết bị truyền thống.
* **6/1/2009:** Microsoft công bố “Project Natal” ở hội nghị thường niên E3.
* **6/13/2010:** Trong suốt hội nghị E3, đổi tên “Project Natal” thành Kinect, chính thức là 1 thiết bị hỗ trợ cho Xbox 360.
* **11/4/2010:** Microsoft chính thức tung ra thị trường Kinect, cũng từ đây, nhưng kế hoạch phát triển Driver nguồn mở cho Kinect của các tổ chức/ hacker cũng bắt đầu thực hiện.
* **11/10/2010:** hacker trẻ tuổi Hector đã phát triển thành công Driver cho Kinect.



Hình . Demo sử dụng kinect trên Mac OS X

* **2/21/2011:** Microsoft lên kế hoạch cho việc phát triển bộ SDK hỗ trợ cho kinect, tuy nhiên đến nay (6/2011), các API hỗ trợ từ thư viện này còn rất sơ khai.

Kinect sử dụng webcam, thiết bị thu phát sóng hồng ngoại, và thiết bị thu nhận âm thanh để ghi nhận tín hiệu chuyển động của game thủ và nhận dạng các lệnh điều khiển thông qua giọng nói, giúp cho các game thủ tương tác với Xbox 360 mà không cần chạm vào bất kì thiết bị điều khiển nào. Trong Hình 3.1, game thủ có thể đấm bốc như thật.

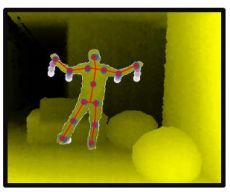
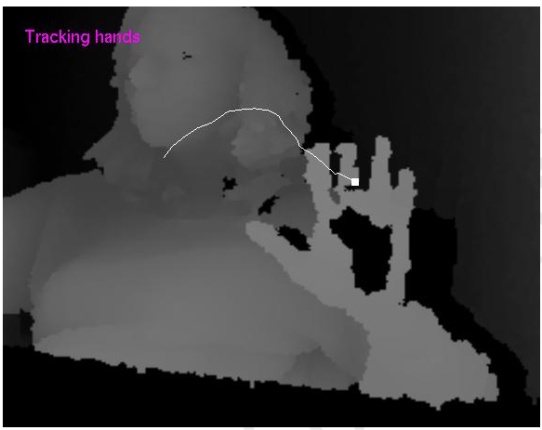


Hình . Nhận diện hành động game thủ và điều khiển nhân vật trong game.[ 2]

Kinect được bán rộng rãi lần đầu tại Bắc Mỹ vào tháng 11 năm 2010 [ 4], sau đó là Úc, New Zealand, Singapore, Nhật Bản và nhanh chóng có mặt trên khắp các cửa hàng đồ chơi trên toàn thế giới. Kinect là 1 thiết bị giải trí tiên tiến, đi tiên phong trong lĩnh vực thực tế ảo, nên hiện đang lập kỉ lục là thiết bị kĩ thuật số được bán chạy nhất trên toàn thế giới.

Kinect, hỗ trợ các chức năng tương tác sau:

* Ghi nhận chuyển động tay (hand gesture), bao gổm các hành động xoay vòng (circle), di chuyển tay (wave gesture), push,...
* Ghi nhận chuyển động toàn cơ thể (full body skeleton), xác định các vị trí chính của cơ thể như đầu, vai, cẳng tay, chân,…
* Điều khiển bằng giọng nói.
* Nhận dạng số người đang chơi
* Phân biệt đối tượng dựa vào độ sâu.



Hình . Các chức năng thường dùng của Kinect

Kinect có cấu tạo thon, dài, nằm ngang bao gồm 1 webcam có thể ghi nhận được hình ảnh; 1 thiết bị phát tia hồng ngoại, 1 thiết bị thu nhận tín hiện hồng ngoại (xem chi tiết trong Hình 3.3), 1 thiết bị ghi nhận âm thanh. Kinect dùng công nghệ xác định khoảng cách dùng camera kết hợp thiết bị thu phát tia hồng ngoại, phát triển bởi các lập trình viên Israel thuộc PrimeSense, có khả năng ghi nhận thông tin 3D của đối tượng dựa trên các thông tin, cấu trúc của các tia hồng ngoại nhận được. Có thể xem như Kinect là 1 3D scanner, xây dựng cấu trúc vật thể 3D dựa trên thông tin ảnh và khoảng cách đến từng điểm của ảnh.

## Các loại gallery online

Quảng cáo sản phẩm 2d, Mô hỉnh 3d

## Mục tiêu

## Nội dung

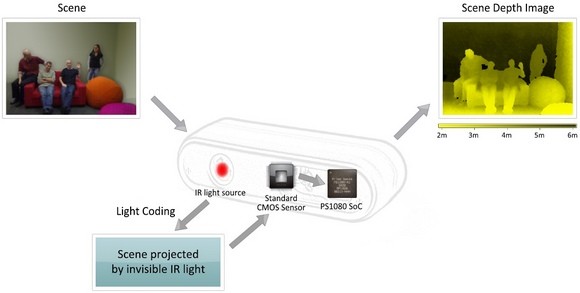
# Một số vấn đề lập trình với kineck

**Tóm tắt chương:**

✍ Nội dung chương này trình bày kĩ thuật lấy thông tin từ kineck, bao gồm thông tin ảnh màu, ảnh độ sâu, và tình trạng bị nhiễu của ảnh đầu vào. Để giải quyết ảnh bị nhiễu, chúng tôi áp dụng kĩ thuật xử lý ảnh bằng bộ lọc Bayer, với thuật toán nội suy màu EdgeAware.

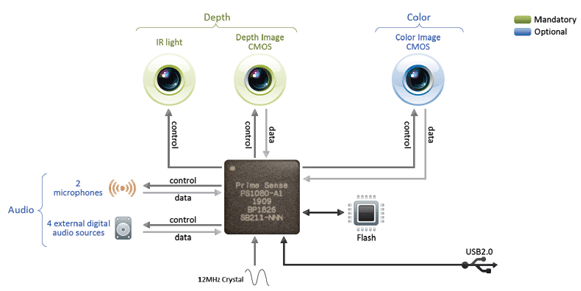
## Thông tin được từ kinect

Để hiện thực hóa 1 đối tượng 3D chúng ta cần xác định ít nhất phải có các thông tin về màu sắc độ sâu từ nhiều điểm trên đối tượng ở nhiều góc nhìn khác nhau. Phần này trình bày cách thức hoạt động của kinect, những thông tin thu nhận được, để có thể trả ra các thông tin về ảnh và độ sâu như mong muốn.



Hình . Cách thức xác định độ sâu từng điểm ảnh và kết quả đạt được[ 7]

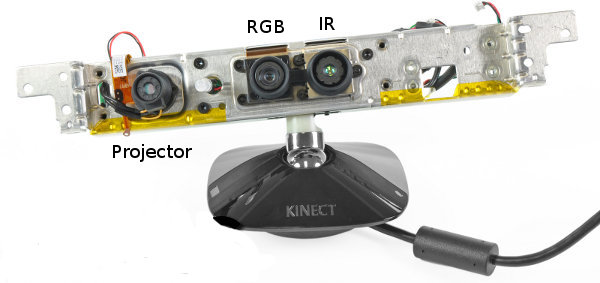
Kinect bao gồm 4 bộ phận chính, bao gồm thiết bị thu tín hiệu âm thanh (tương tự microphone), thiết bị cảm biến thu nhận hình ảnh, thiết bị thu phát sóng hồng ngoại. Tuy nhiên trong phạm vi luận văn này, chúng em bỏ qua chi tiết thiết bị ghi nhận âm thanh, tập trung vào các vấn đề liện quan đến ảnh RGB và ảnh độ sâu. Để thu nhận hình ảnh, sử dụng thiết bị thu nhận hình ảnh là 1 webcam bình thường. Để thu nhận độ sâu từng điểm ảnh, Kinect hoạt động dựa trên công nghệ Light Coding. Công nghệ LightCoding hoạt động dựa trên việc phát ra chùm tia hồng ngoại đặc trưng riêng từng tia (không thấy được dưới mắt thường). Kinect dùng 1 bộ cảm biến CMOS chuẩn, để ghi nhận lại các tia hồng ngoại bị phản xạ lại khi tiếp xúc với môi trường, dựa vào các đặc trưng mà xác định cụ thể vị trí tia hồng ngoại trong chùm tia và độ sâu của tia đo được. Sử dụng PS1080 SoC chip tính toán song song để xác định độ sâu của toàn bộ chùm tia phản xạ và xuất ra độ sâu của tất cả điểm ảnh [ 7]. Phương pháp này có thể chống được nhiễu của ánh sáng phản chiếu trong phòng (ambient light) (Chi tiết về cách hoạt động vận hành của Chip này xin xem trong Hình 3.2).



Hình . Bộ xử lý trung tâm của kinect[ 7]

Hình mình họa cơ bản các tương tác giữa bộ xử lý trung tâm của kinect và các thành phần khác:

* Tương tác điều khiển và lấy dữ liệu từ các thiết bị đầu cuối (Depth, Color, Audio).
* Tương tác với bộ nhớ vật lý (Flash).
* Tương tác với các thiết bị bên ngoài thông qua cổng USB (Xbox 360, máy vi tính,…).



Hình . Hình minh họa cấu tạo bên trong Kinect

Thông số kĩ thuật cơ bản của Kinect:

* Thiết bị cảm biến ảnh màu của Kinect có thể ghi nhận ảnh màu RGB (8 bit) với tần số 30Hz, kích thước khung hình 640 x 480 điểm ảnh.
* Thiết bị ghi nhận độ sâu cũng có kích thước khung hình là 640 x 480 điểm, mổi điểm độ sâu có 11 bit, do đó có thể phân biệt được 2048 độ sâu khác nhau trong cùng 1 ảnh độ sâu. Sử dụng với Xbox, thông thường giới hạn nhận biết độ sâu là 1.2 – 3.5m, có thể mở rộng 0.7 – 6m. Góc mở theo phương ngang là 580, theo phương dọc là 400[ 7], với khoảng cách ngắn nhất, có thể đạt tỉ lệ 1.3mm tương đương 1 pixel.
* Thiết bị ghi nhận âm thanh 16 bit, tần số 16HZz.
* Hỗ trợ cỗng USB nên hiện nay có thể kết nối với nhiều thiết bị, trong đó có máy vi tính.

## Cấu hình Kinect trên Windows

Kinect là 1 thiết bị vốn chuyên dùng cho Xbox 360, hỗ trợ tiếp nhận điều khiển từ người dùng mà không cần chạm vào thiết bị. Tuy nhiên, để sử dụng và lập trình với kinect trên phần cứng khác thì không dễ dàng, cụ thể trong trường hợp này là sử dụng và lập trình kinect trênWindows. Hướng giải quyết vấn đề này là cài đặt Driver cho thiết bị, cài đặt 1 thư viện hỗ trợ các tương tác với kinect và chạy thử 1 chương trình lấy thông tin bất kì để kiểm tra khả năng vận hành của kinect trên Windows. Chúng em sử dụng driver Alvin 2 Kinect Sensor và thư viện nguồn mở OpenNI. Các bước cài đặt cơ bản như Hình 3.1.

Hình . Các bước cơ bản thực hiện cài đặt Driver kinect và thư viện OpenNI

Trước khi cài đặt, chúng ta cần chuẩn bị Driver cho kinect, các file cài đặt thư viện OpenNI, file xml cấu hình riêng cho kinect.

Bước 0: Xóa các driver cũ hoàn toàn khỏi máy tính, đã cài trước đó có liên quan đến kinect. Đây là bước đơn giản, nhưng thường bị bỏ qua khi cài đi cài lại nhiêu lần, có thể khiến kinect tự động nhận lại driver cũ không còn tương thích với hệ thống.

Bước 1: Cài đặt Driver Alvin Kinect Sensor, chọn platform làWin 32 hay 64 bit.

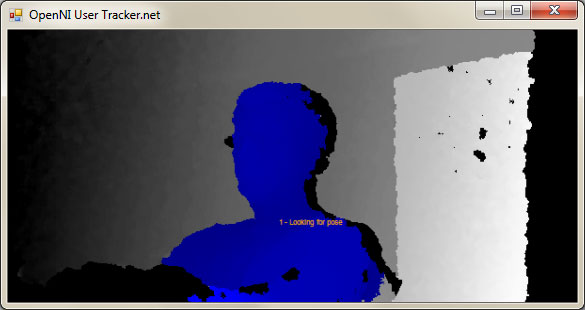
Bước 2: Cài đặt OpenNI binaries, chọn bản Stable hoặc bản Unstable đều được, tùy thuộc hệ điều hành mà bạn chọn phiên bản cài đặt cho x86 hay x64, và dùng cho lập trình viên (Dev).

Bước 3: Cài đặt OpenNI Compliant Middleware Binaries, dù hệ điều hành đang dùng là x86 hay x64 thì nên dùng phiên bản cho x86 để đảm bảo tính ổn định, phiên bản cho x64 chạy thiếu ổn định.

Bước 4: Cài đặt OpenNI Compliant Hardware Binaries tương tự bước 3.

Bước 5: Copy 3 file cấu hình Sample-Scene.xml, Sample-Tracking.xml, Sample-User.xml vào thư mục “c:\Program Files (x86)\PrimeSense\NITE\Data\” và file SamplesConfig.xml vào thư mục “c:\Program Files (x86)\OpenNI\Data\”, mục đích để cấu hình kinect xuất ra những định dạng thông tin như ảnh màu RGB, depth, tần số, kích thước khung hình,…

Kiểm tra kinect đã hoạt động được chưa, ta cắm kinect qua cổng usb, và kích hoạt 1 chương trình demo trong thư mục “c:\Program Files (x86)\OpenNI\Samples\Bin\” hoặc thư mục “c:\Program Files (x86)\PrimeSense\NITE\Samples\Bin\”.



Hình . Chương trình demo sử dụng được kinect trên Windows.

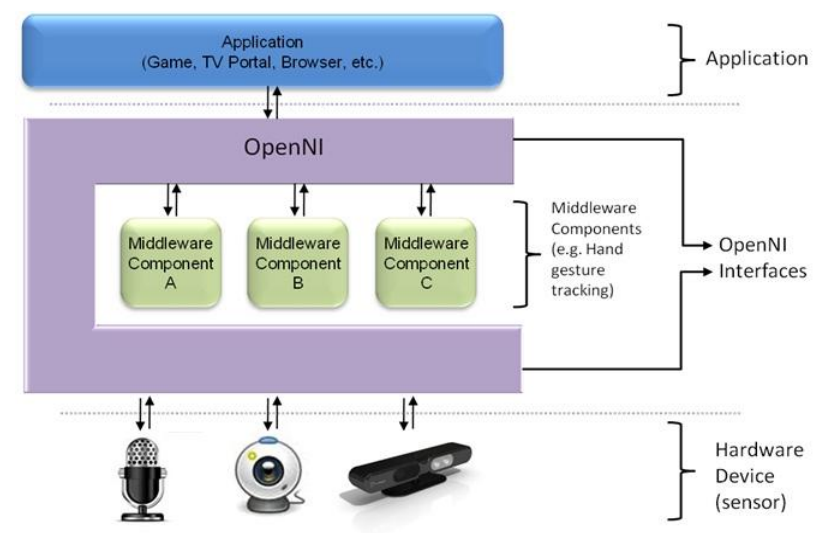
## Lấy thông tin ảnh RGB và ảnh độ sâu theo thời gian

Để lấy các thông tin cần thiết từ kinect, ta không thể tương tác trực tiếp với driver của thiết bị, việc này đòi hỏi kĩ thuật cao và khả năng lập trình rất tốt và am hiểu cấu tạo thiết bị. Hướng giải quyết là dùng 1 thư viện được cộng đồng nguồn mở hỗ trợ, để lấy dữ liệu 1 cách dễ dàng và được chuẩn hóa. Thư viện OpenNI cung cấp 1 giải pháp khá toàn diện cho vấn đề thao tác với dữ liệu từ kinect.

### Các thành phần cơ bản của OpenNI

Thư viện OpenNI có 3 thành phần chính:

* Tương tác với Application: đóng gói các thành phần trong OpenNI, cung cấp cho lập trình viên các API thao tác với dữ liệu đã chuẩn hóa và dễ dàng sử dụng.
* Tương Tác với MiddleWare: tương tác với các thành phần xử lý dữ liệu, phục vụ cho nhiều mục đích khác nhau như: phân tích cử động cơ thể người, phân tích thao tác tay, phân tích phông nền, xác định nền nhà,…
* Tương tác với phần cứng: Chuẩn hóa các tương tác với nhiều loại phần cứng thông qua dùng các file cấu hình động xml.



Hình . Các thànhphần cơ bản của OpenNI.

Các chức năng nổi trội mà OpenNi hiện đang hỗ trợ lập trình viên thao tác với kinect:

* Alternative View: Do, mỗi bộ cảm biến ghi nhận thông tin ảnh và thông tin IR ở 2 vị trí khác nhau trên kinect, nên khi lấy dữ liệu ảnh và độ sâu, sẽ không khớp với nhau về góc nhìn, OpenNI dựa trên vị trí cố định giữa projector và webcam trên kinect để ánh xạ dữ liệu ảnh, độ sâu, IR vào cùng 1 hệ trục tọa độ.
* Cropping: Hỗ trợ cắt bớt dữ liệu xuất ra, thay vì phải lấy toàn bộ khung hình 640 x 480, thì chỉ lấy giới hạn kích thước để dữ liệu kết quả nhỏ gọn hơn, phù hợp cho những ứng dụng đòi hỏi chạy real time.
* Fame Sync: Khi lấy 1 thông tin ảnh hoặc độ sâu, thì không cần quan tâm vấn đề đồng bộ dữ liệu giữa thành phần này, nhưng khi cần lấy 2 thông tin này cùng lúc, đòi hỏi 1 cơ chế giúp đồi bộ quá trình ghi nhận dữ liệu ảnh RGb và độ sâu.
* Mirror: Cơ chế ánh xạ từ trái sang phải và ngược lại để hình ảnh thu được không bị ngược chiều so với thực tế (áp dụng chủ yếu cho việc nhận dạng chuyển động tay, nếu không có cơ chế này thì có thể nhận nhầm từ tay trái sang tay phải.
* Pose and User Detection: Giúp xác định vị trí của người khi di chuyển vào góc nhìn của kinect.
* Skeleton: Hỗ trợ xuất ra thông tin khung xương của đối tượng (người).
* Error State: cung cấp tình trạng dữ liệu đc lấy ra, hoặc kiểm tra các Node có tồn tại hay không.
* LockAware: chia sẽ kinect giữa các phần mềm.
* Recording andPlaying: ghi nhận thông tin trực tiếp từ kinect vào fiel .ONI (định dạng riêng của OpenNI), hỗ trợ replay lại bằng cách đọc từ file .ONI mà không cần chỉnh sửa lại cấu hình các thành phần bên trong.

Để chuẩn hóa việc lấy dữ liệu thô lẫn đã xử lý, OpenNI định nghĩa 1 thành phần là Production Node. Mỗi loại Production Node được cung cấp 1 số hàm để rút trích dữ liệu. Một Production Node, có thể lấy dữ liệu trực tiếp từ thiết bị (Image Generate, Depth Generate, IR generate, Audio generate), hay lấy dữ liệu từ các Production Node cấp thấp hơn (Gestures Alert Generate, Scene Analyzer, Hand Point generator). Tham khảo thêm về công dụng các loại Node này tại đây [ 6]. Quá trình lấy dữ liệu từ các ProductionNode cấp thấp, rồi tổng hợp, phân tích dữ liệu ở những Production Node cấp cao hơn gọi là Production Chain. Trong các loại Production Node đã liệt kê, chúng em chỉ qua tâm 2 Production Node chính là Image Genarate và Depth Genarate. Các bước để lấy dữ liệu từ kinect thông qua 2 Production Node Image Generate và Depth Genarate (xemHình 3.4).

Hình . Các bước lấy dữ liệu từ kinect

Bước 1: Khởi tạo biến context, biến này có nhiệm vụ tương tác trực tiếp với kinect. Thiết lập cấu hình thư viện OpenNI tương thích với kinect thông qua các thông tin cấu hình trong file xml. Kiểm tra kinect có hỗ trợ trả về các thông tin như ảnh RGB và thông tin độ sâu hay không.

1 đoạn trong file cấu hình xml:

<ProductionNodes>

<Node type="Image" name="Image1">

<Configuration>

<MapOutputMode xRes="640" yRes="480" FPS="30"/>

<Mirror on="true"/>

</Configuration>

</Node>

<Node type="Depth" name="Depth1">

<Configuration>

<MapOutputMode xRes="640" yRes="480" FPS="30"/>

<Mirror on="true"/>

</Configuration>

</Node>

</ProductionNodes>

Khởi tạo thông tin cho biến context, kiểm tra kinect:

xn::Context m\_ni\_context;

XnStatus status = m\_ni\_context.InitFromXmlFile(config\_file, &errors);

xn::DepthGenerator m\_ni\_depth\_generator;

xn::ImageGenerator m\_ni\_rgb\_generator;

status = m\_ni\_context.FindExistingNode(XN\_NODE\_TYPE\_DEPTH, m\_ni\_depth\_generator);

check\_error(status, "Find depth generator");

status = m\_ni\_context.FindExistingNode(XN\_NODE\_TYPE\_IMAGE, m\_ni\_rgb\_generator);

check\_error(status, "Find image generator");

Bước 2: Bổ sung các thiết lập bổ sung, hiện tại OpenNI hỗ trợ thiết lập thêm các thông tin sau để hỗ trợ quá trình lấy dữ liệu từ kinect tốt hơn, trong các chức năng OpenNI cung cấp, hiện chúng em chỉ sử dụng chức năng đồng bộ hóa dữ liệu ảnh và ảnh độ sâu (chi tiết trình bày trong bước 4); thiết lập ảnh RGB và ảnh độ sâu chung 1 góc nhìn (xem Hình 3.5, các vùng có màu đen là không có thông tin độ sâu, ta nhận thấy vùng màu đen nằm ở rìa ảnh, đã qua quá trình xử lý đề chuyển các điểm có độ sâu đúng vị trí với các điểm màu RGB).

m\_ni\_depth\_generator.GetAlternativeViewPointCap().SetViewPoint(m\_ni\_rgb\_generator);

Bước 3: Kích hoạt quá trình lấy dữ liệu từ kinect, có thể kiểm tra status để xem tình trạng của quá trình lấy dữ liệu.

status = m\_ni\_context.StartGeneratingAll();

Bước 4: Đợi đến khi có dữ liệu, cập nhật dữ liệu vào từng Node. Trong OpenNI hỗ trợ nhiều cơ chế để lấy cùng lúc nhiều loại dữ liệu, trong trường hợp này là lấy ảnh RGB và ảnh độ sâu.

xn::Context::WaitAnyUpdateAll(): Chỉ cần 1 trong nhiều Node có dữ liệu, lập tức gửi tín hiệu cập nhật cho chương trình.

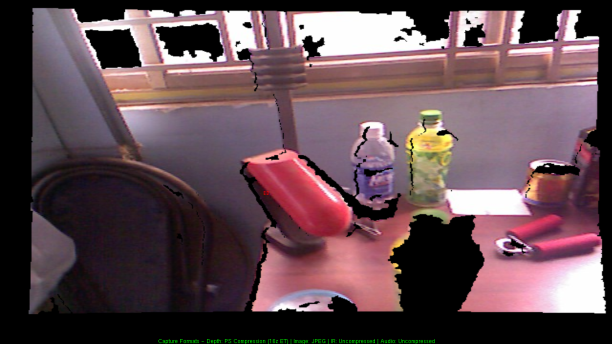
xn::Context::WaitOneUpdateAll(): Chỉ định 1 ProductioNode, nếu Node này ó dữ liệu mới, gửi tín hiện cập nhật cho chương trình.

xn::Context::WaitNoneUpdateAll(): Lấy dữ liệu mà không cần quan tâm dữ liệu cũ hay mới.

xn::Context::WaitAndUpdateAll(): Đợi đến khi tất cả các Node đều có dữ liệu mới, rồi mới gửi tín hiệu cập nhật.

Trong chương trình, ban đầu, thông tin giữa ảnh và độ sâu không khớp với nhau, ảnh cập nhật 1-2 frame rồi độ sâu mới cập nhật, hoặc ngược lại, tuy nhiên cũng không theo bất kì quy tắc nào, có thể thay đổi quy tắc từ độ sâu cập nhật trước sang ảnh RGB cập nhật dữ liệu trước một cách ngẫu nhiên và liên tục. Để giải quyết hiện tượng trên, chúng em sử dụng xn::Context::WaitAndUpdateAll() để đợi đến khi có dữ liệu mới từ 2 Production Node là ImageGenarate và DepthGenerate rồi mới lấy dữ liệu ra, như vậy đảm bảo dữ liệu nhận được sẽ khớp với nhau.

m\_ni\_context.WaitAndUpdateAll();



Hình . Minh họa trước và sau đồng bộ hóa dữ liệu ảnh và độ sâu.

Dựa vào Hình 3.5, ta dễ dàng nhận thấy, đối với các ảnh không đồng bộ giữa ảnh màu RGB và ảnh độ sâu, những chỗ không nhận được độ sâu (thường nằm ở mép, rìa đối tượng), sẽ bị lệch sang vị trí khác.

Bước 5: Lấy dữ liệu và xử lý, đây là giai đoạn đã chuyển giao dữ liệu nhận được cho chương trình, sau khi xử lý xong, quay lại bước 4 để tiếp tục lấy dữ liệu từ kinect đến khi kết thúc chương trình.

xn::DepthMetaData depthMD;

xn::ImageMetaData rgbMD;

m\_ni\_depth\_generator.GetMetaData(depthMD);

m\_ni\_rgb\_generator.GetMetaData(rgbMD);

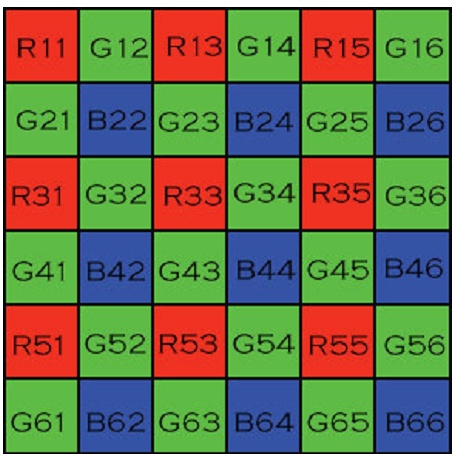
const XnDepthPixel\* pDepth = depthMD.Data();

const XnImagePixel\* pRgb = rgbMD.Data();

## khử nhiễu ảnh màu

Trong quá trình lấy dữ liệu ảnh màu RGB và ảnh độ sâu, chúng tôi nhận thấy ảnh màu thu được không tốt như các webcam thông thường (cùng kích thước và cùng cảnh chụp), ảnh RGB bị hiện tượng nhiễu (có sọc đỏ tương tự nhau), rõ nét nhất là ở các vị trí đường biên của các đối tượng trong ảnh RGB. Vấn đề này xuất hiện hầu hết trong các webcam hoặc máy chụp hình kĩ thuật số hiện nay do sử dụng bộ cảm biến đơn sắc (chỉ nhận 1 trong 3 màu), 3 bộ cảm biến liên tiếp ghi nhận thông tin màu của 1 ví trí trên đối tượng [ 1]. Điều này ảnh hưởng đến ảnh kết quả, độ chính xác của các điểm đặc trưng trong ảnh RGB, được tiếp tục dùng trong thuật toán tính so khớp giữa các ảnh. Để giải quyết vấn đề này, một kĩ thuật thường được sử dụng là áp dụng 1 hay vài bộ lọc phù hợp để làm giảm hiện tượng nhiễu. Bộ lọc phù hợp trong trường hợp này là bộ lọc Bayer [ 3].

Nguyên lý hoạt động của Bayer rất đơn giản, áp dụng 1 mặt nạ màu (Bayer Pattern) phù hợp lên từng điểm ảnh, đổng thời thực hiện thuật toán nội suy màu căn cứ trên các điểm lân [ 2][ 4]. Nhiệm vụ chính của Bayer Pattern chính là đảm bảo thuật toán tính nội suy màu, ứng với từng kênh màu (channel) chỉ quan tâm những màu tương ứng trong các điểm ảnh lân cận. Như vậy, ta sẽ có 4 vị trí khác nhau, mà thuật toán nội suy màu cần quan tâm. Quy ước, R là màu đỏ, G là màu xanh lá, B là xanh dương, ta có các vị trí cụ thể như sau: vị trí R, vị trí G dòng R, vị trí G dòng B, vị trí B.



Hình . mặt nạ màu thông dụng của bộ lọc Bayer[ 2][ 4]

Trong bộ lọc Bayer, để tính toán màu cho từng điểm ảnh, có thể sử dụng các thuật toán nội suy phổ biến như Pixel Doubling, Bilinear, Gradient Based, High Quality Linear [ 2]. Nhiệm vụ chính lúc này là giảm sọc ngang trên ảnh, chúng tôi chọn thuật toán nội suy EdgeAware.

Các bước áp dụng bộ lọc Bayer - nội suy EdgeAware như sau:

Bước 1: Với mỗi điểm ảnh, xác định vị trí ứng với mặt nạ màu (1 trong 4 vị trí đã nêu ở phần trên).

Bước 2: Với từng vị trí, từng kênh màu (channel) cụ thể R, G, hay B, ta tính nội suy màu theo EdgeAware như sau:

Vị trí G dòng R, Vị trí G dòng B: Với từng kênh màu, tính trung bình của các điểm lân cận có mặt nạ màu tương ứng.

Vị trí R, Vị trí B: Với kênh màu R, B, tính trung bình của các điểm lân cận có mặt nạ màu tương ứng. Với kênh màu G, xác định 2 điểm G theo chiều dọc hay chiều ngang có độ chênh lệch G ít nhất, tính trung bình cộng 2 điểm G đó. Nếu độ chênh lệch bằng nhau, tính trung bình cộng 4 điểm G[ 5]. Đây chính là nguyên nhân chúng tôi chọn thuật toán nội suy EdgeAware thay vì chọn các thuật toán khác. Đối với các sọc ngang độ biến màu theo chiều ngang cao, do đó, nội suy màu sẽ thường sử dụng thông tin của các điểm màu lân cận theo chiều dọc



Hình . Ảnh màu (được phóng to 4 lần) trước và sau khi sử dụng bộ lọc Bayer (thuật toán nội suy EdgeAware)

Kết quả đạt được sau khi áp dụng bộ lọc Bayer, ảnh thu được từ kineck giảm hẳn hiện tượng sọc ngang. Lưu ý, thu ật to án này cần xử lý riêng các điểm màu ở mép ảnh (ở cạnh và góc ảnh) vì tại các vị trí này thiếu thông tin các điểm màu lân cận.

# Ánh xạ thông tin nhận được sang đám mây điểm

## Tổng quan

Các thông tin thu được từ kineck còn rời rạc, cần chuẩn hóa để sử dụng.

## Các thông số cơ bản của Camera

Ma trận intrinsic, ma trận xoay, tịnh tiến, hệ trục tọa độ camera

## Kĩ thuật chuyển ảnh RGB và ảnh độ sâu sang đám mây điểm

Phép chiếu trong không gian

Xử lý vấn đề chuyển đổi hệ trục để tính toán trong Opencv

Tính các thông tin của ma trận intrinsic (fx, fy, cx, cy) dùng hàm estimateCalibratrion,

+tính đơn vị trong thế giới thực, tương ứng với bao nhiêu pixel

+real world (0, 0, -1) --> điểm chiếu trong image --> điểm (cx, cy)

+real world (1, 1, -1) --> điểm nằm ở biên ảnh --> lấy chênh lệch với điểm trên ra (fx, fy)

+điểu chỉnh 4 đối số này bằng các tham số mặc định (ko biết căn cứ vào đâu)

Áp dụng công thức sau:

+từ 3d sang 2d: point2d = intrinsics \* rt \* point3d

+từ 2d sang 3d: point3d = reverse(intrinsics \* rt) \* point2d

+thực tế trong code thì công thức trên phức tạp hơn: intrinsics \* rt tương đương intrinsics \* projection \* to\_opencv \* camera\_transform;

trong đó: projection = I; to\_opencv: ma trận giúp thay đổi hệ trục dùng trong opencv; camera\_transform: ma trận rt

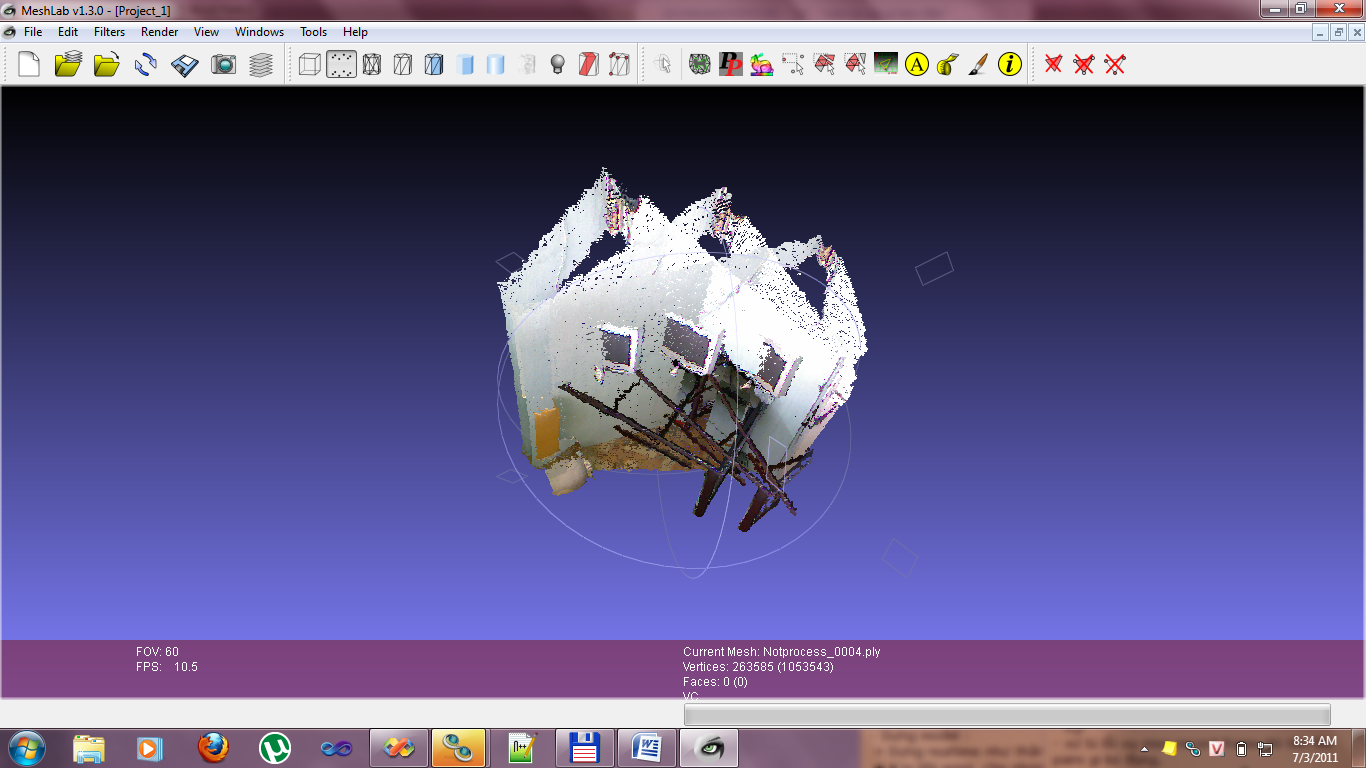
# Mô hình hóa thế giới thực

**Tóm tắt chương:**

✍ Nội dung chương này trình bày kĩ thuật mô hình hóa thế giới thực từ kineck, các vấn đề gặp phải và hướng giải quyết. Phần quan trọng nhất trong chương này là cách xácđịnh vị trí tương đối giữa 2 frame (2 đám mây) trong không gian 3D.

## Tổng quan

Trong thế giới thực, để quan sát toàn bộ đối tượng, người ta thường quan sát với nhiều góc nhìn, mỗi góc nhìn tương ứng với vị trí quan sát khác nhau. Xét đối với kineck, tương ứng với việc đặt kineck ở các ví trí khác nhau để ghi nhận cùng 1 đối tượng. Tuy nhiên, dữ liệu thu nhận được từ kineck lại rời rạc, mỗi lần thực hiện chỉ ghi nhận được thông tin ảnh màu RGB và ảnh độ sâu tại ví trí hiện tại (xem mỗi lần lấy thông tin ảnh RGB và ảnh độ sâu như 1 khung hình (frame)). Khi ánh xạ vị trí của các điểm ảnh này trong không gian 3D, tức là ánh xạ lên không gian 3D tương ứng với vị trí đặt kineck, hiểu theo hướng khác là lấy hệ trục tọa độ của kineck làm gốc (hệ trục camera). Khi có nhiều hơn 1 frame, thì các điểm được ánh xạ lên không gian 3D sẽ bị chồng chéo lên nhau.



Hình .Minh họa 3 frame trên cùng 1 hệ trục của kineck.

Với tình trạng như Hình 3.1, chúng ta cần chuyển từng frame lên hệ trục tọa độ trong thế giới thực. Nhưng vấn đề đặt ra ở đây là, với từng frame rời rạc thu được (thiếu thông tin vị trí kineck trong thế giới thực), thì làm thế nào để chuyển sang hệ trục tọa độ thế giới thực 1 cách chính xác ?? Chúng tôi đề nghị 1 giải pháp là xác định mối tương quan giữa 2 frame trên ảnh RGB (2D), từ đó, xác định vị trí tương đối (Pose) của 2 frame trong không gian.

## Quy trình thực hiện

Trong phần này, chúng tôi trình bày các bước cơ bản giải quyết vấn đề mô hình hóa thế giới thực, kĩ thuật chi tiết sẽ được trình bày ở mục xx.

Bước 1: Chọn 1 frame làm gốc, ánh xạ frame sang đám mây điểm trong x gian 3D. Gán vị trí trong thế giới thực trong không gian này, có thể xem như hệ trục tọa độ của kineck trùng với hệ trục tọa độ trong thế giới thực.

Bước 2: Khi xuất hiện frame mới, kiểm tra với tất cả các frame cũ, xem frame nào có nhiều điểm tương đồng (2D) nhất, hoặc đủ số điểm tương đồng cần thiết. Kĩ thuật này được trình bày chi tiết trong mục xx.

Bước 3: Với mỗi cặp điểm tương đồng trên 2D này, loại bỏ các cặp không có độ sâu, ánh xạ sang các cặp trong không gian 3D, lọc bớt các cặp không chính xác so với các cặp khác.

Bước 4: xác định vị trí tương đối giữa 2 frame (đã được chuyển sang 2 đám mây) trong không gian.

## Kĩ thuật xác định các cặp điểm tương đồng giữa 2 ảnh.

SURF/SIFT…

## Kĩ thuật lọc bỏ các cặp điểm không chính xác

## Kĩ thuật xác định vị trí 3D (Pose) tương đối giữa 2 đám mây.

Trong hầu hết các ứng dụng, thư viện ghép nối đám mây hiện nay (chương trình meshlab – plugins align, thư viện pcl – code align, rgbdemo của Ph.D Burrus Nicolas), qua phân tích code của các chương trình/ thư viện này, kĩ thuật thường được sử dụng là kết hợp kĩ thuật chọn cặp điểm ngẫu nhiên (ransac) và thuật toán tối ưu hoá hàm biến đổi giữa 2 đám mây (Levenberg–Marquardt). Vị trí tương đối giữa 2 đám mây, về bản chất gồm 2 thành phần cơ bản là ma trận xoay và ma trận tịnh tiến. Do đó, nhiệm vụ chính trong phần này là xác định được ma trận xoay và ma trận tịnh tiến để biến đổi đám mây này sang đám mây khác, mà độ lỗi (khoảng cách giữa cặp điểm tương đồng giữa 2 đám mây sau biến đổi) là tối ưu nhất.

### Tối ưu hóa ma trận biến đổi giữa 2 đám mây điểm

Thuật toán Levenberg–Marquardt tương tự như các thuật toán tối ưu khác, là 1 thuật toán lặp, bắt đầu từ tham số khởi tạo với độ lỗi tính toán được cao, sau mỗi lần lặp đỗ lỗi giảm dần dựa trên việc chọn 1 tham số ước lượng phù hợp hơn (damping parameter) [ 6]. Các thông tin cần cung cấp cho thuật toán bao gồm: các cặp điểm tương đồng (xi, yi), hàm biến đổi f từ x sang y, tham số và cách xác định độ lỗi S sau khi biến đổi để công thức  đạt tối thiểu [ 7].

Áp dụng trong trường hợp cần tính vị trí tương đối giữa 2 đám mây, thì các thông tin này được ánh xạ như sau:

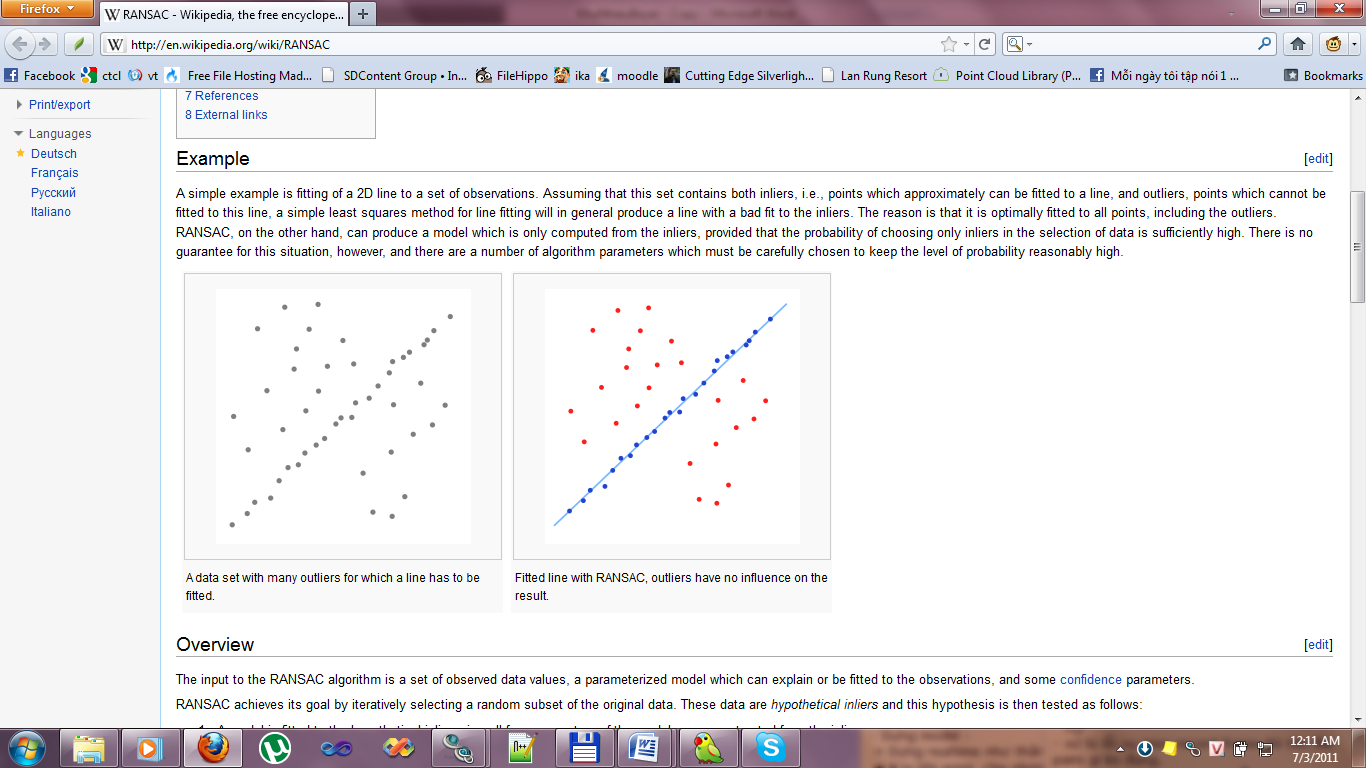
* Cặp điểm tương đồng (xi, yi): là các cặp điểm tương đồng trên ảnh RGB, được xác định bằng thuật toán tìm đặc trưng ảnh (xem mục xx) và ánh xạ thành điểm trong không gian 3D, đã được lọc bỏ bớt các điểm không xác định được chiều sâu hoặc không chính xác (xem mục xx).
* Hàm biến đổi f từ điềm x sang điểm y’: Là phép chiếu (tịnh tiến, xoay) trong không gian 3D, từ điểm x sang y. Đối số là ma trận xoay, ma trận tịnh tiến và điểm x. Kết quả trả ra là điểm y’ tương ứng.
* Tham số : chính là ma trận xoay và tịnh tiến trong không gian 3D, cũng chính là biến số cần tối ưu trong thuật toán này.
* Cách xác định độ lỗi S: là hàm tính bình phương khoảng cách trong không gian 3D giữa 2 điểm y và điểm y’ (x sau biến đổi).

Diễn giải thuật toán 1 cách đơn giản là tìm 1 ma trận biến đổi (tịnh tiến, xoay) sao cho có thể biến đổi đám mây điểm (chứa điểm xi) sang đám mây điểm (chứa điểm yi) một cách chính xác nhất có thể được.

### Xác định ma trận biến đổi tốt nhất

Trong số các cặp điểm tương đồng 3D tính được, không thể tránh khỏi việc có một số cặp điểm không chính xác (sau khi xác định cặp điểm tương đồng trên ảnh RGB, ánh xạ lên không gian 3D, 1 số điểm có hiện tượng ở đám mây A thì nằm trên đường biên đối tượng, ở đám mây B thì điểm tương ứng nằm trên tường) hoặc có độ sai số nhất định so với vật thật, dẫn đến khoảng cách, hướng trong không gian 3D của các cặp này, ảnh hưởng đến kết quả tính ma trận biến đổi trong thuật toán ở phần trước. Giải pháp thường xuyên được áp dụng để giải quyết vấn đề này là dùng thuật toán Ransac.

Thuật toán Ransac (RANdom SAmple Consensus), được công bố bởi 2 nhà thống kê đại tài là Fischler and Bolles, là thuật toán lặp lấy tập mẫu ngẫu nhiên từ tập nguồn có chứa nhiều phần tử lỗi (outlier), để việc tính toán trên tập nguồn tránh những phần tử lỗi (outlier)[ 8]. Thuật toán Ransac được đánh giá là 1 thuật toán mạnh mẽ, do thay vì dùng số lượng điểm nhiều nhất có thể, Ransac lại chọn số điểm ít nhất mà vẫn đạt hiệu quả tương đương[ 9]. Hai vấn đề cần quan tâm khi sử dụng Ransac là: tập mẫu tối thiểu bao nhiêu là phù hợp và lặp bao nhiêu lần thì kết quả đạt được là đủ tốt. Đối với mỗi lần lặp, số lượng phần tử tối thiểu và số lần lặp cần thiết sẽ được tính lại dựa trên tỉ lệ các phần tử phù hợp (inlier) và các phần tử lỗi (outlier) của những vòng lặp trước đó. Công thức chi tiết tham khảo thêm trong tài liệu gốc [ 10].



Hình . Minh họa việc dùng Ransac áp dụng cho việc tìm đường thẳng trong mặt phẳng[ 8].

Áp dụng thuật toán Ransac cho bài toán 1 số cặp điểm tương đồng trong không gian bị lỗi, ta thực hiện các bước như sau[ 9]:

Bước 1: Chọn ngẫu nhiên 1 số lượng phần tử theo công thức [ 10].

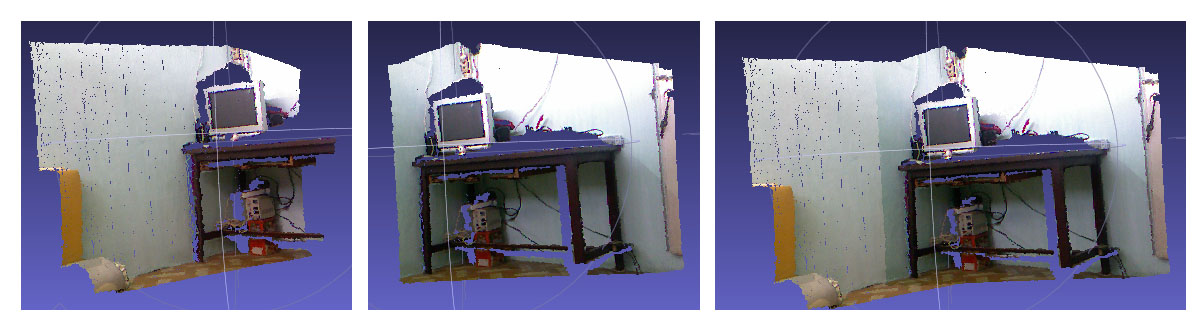
Bước 2: Ước lượng ma trận biến đổi dựa trên thuật toán Levenberg–Marquardt, xác định.

Bước 3: Loại bỏ nếu độ chính xác không chấp nhận được.

Bước 4: Nếu kết quả ở bước 2 chấp nhận được, xác định tỉ lệ các cặp điểm bị lỗi (outlier) khi áp dụng kết quả cho toàn bộ tập nguồn.

Bước 5: Chọn kết quả nàu nếu như tỉ lệ lỗi thấp nhất so với các lần lặp trước đó.

Bước 6: Thực hiện lặp lại bước 1 đến khi hết số vòng lặp cần thiết.



Hình . Minh họa kết quả ghép 2 đám mây sau khi áp dụng ransac và Levenberg–Marquardt để xác định ma trận biến đổi.

# Kỹ thuật trình diễn cảnh 3D trên Silverlight 5

**Tóm tắt chương:**

*✍Nội dung của chương 2 là trình bày kỹ thuật trình diễn cảnh 3D trên môi trường Silverlight 5. Trong chương này, chúng tôi sẽ giới thiệu tổng quan về XNA Graphics và cách vẽ cảnh 3D trên môi trường Silverlight.*

## Giới thiệu

Microsoft Silverlight là một bộ khung ứng dụng (application framework) để phát triển và thực thi các ứng dụng RIA (Rich Internet Application) (theo <http://en.wikipedia.org/wiki/Microsoft_Silverlight>).

Kể từ khi phát hành phiên bản đầu tiên vào tháng tư năm 2007, Silverlight đã không ngừng được cải tiến về chức năng lẫn tốc độ. Theo dự kiến Microsoft sẽ phát hành chính thức phiên bản thứ 5 vào nửa sau năm 2011.

Hình . Kiến trúc Silverlight

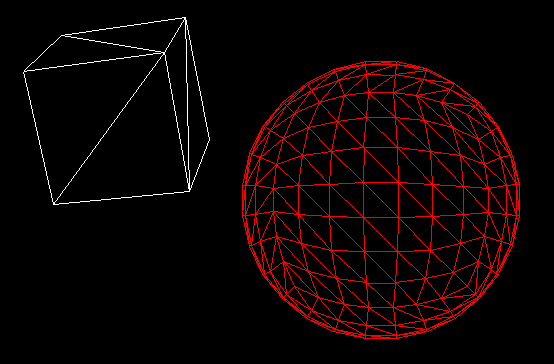
<http://msdn.microsoft.com/en-us/library/bb404713.aspx>

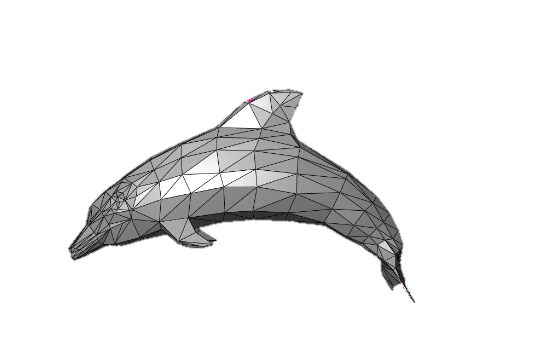
Ngày 13/4/2011, Microsoft đã phát hành phiên bản Silverlight 5 Beta (<http://silverlightvn.net/post/2011/04/22/Silverlight-5-Beta.aspx>) với cải tiến về hiệu năng lẫn tốc độ cùng hơn 40 tính năng mới so với phiên bản trước. Việc tích hợp phần lõi của nền tảng XNA đã khiến Silverlight trở nên mạnh mẽ hơn bao giờ hết với XNA Graphics (còn được gọi là Silverlight 3D), Shader Effect, Sound Effect và quan trọng nhất là khả năng khai thác sức mạnh của GPU (GPU accelerated) ngay trên nền Web.

Trong chương này, chúng tôi sẽ giới thiệu cơ bản về XNA Graphics trong Silverlight 5 và các vấn đề gặp phải khi trình diễn cảnh 3D trên Silverlight cùng giải pháp trong các mục sau.

## Tổng quan về XNA Graphics trong Silverlight 5

XNA Graphics mang lại cho Silverlight khả năng vẽ các đối tượng hình học và các mô hình 3D một cách đơn giản, nhanh chóng cùng với khả năng dễ dàng tùy biến với các phép biến đổi ma trận từ XNA Math và XNA Effects.





<http://channel9.msdn.com/Events/MIX/MIX11/MED06>

Ba thành phần XNA Graphics, Effects và Math được gắn kết vào Silverlight thông qua một control mới là Drawing Surface. Drawing Surface sẽ đóng vai trò là cầu nối giữa nền tảng hiện có của Silverlight với các thành phần đồ họa của XNA (*hình*). Khác với các control khác của Silverlight, tiến trình vẽ của Drawing Surface được phân thành nhiều giai đoạn. Chúng tôi sẽ trình bày sơ lược các giai đoạn trong các mục sau.

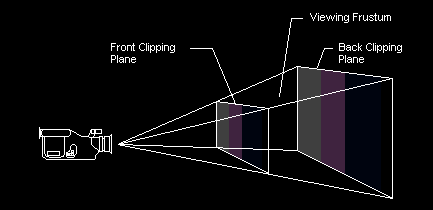
### Vertex Data

Mọi đối tượng đều được định nghĩa bằng một tập hợp các đỉnh (Vertex). Thông tin tại mỗi đỉnh bao gồm tối thiểu tọa độ của đỉnh đó trong không gian và có thể có thêm các thông tin khác như màu sắc, vector vuông góc (normal vector), …

Tất cả các đỉnh của đối tượng đều lần lượt được gửi sang bước hai Vertex Processing.

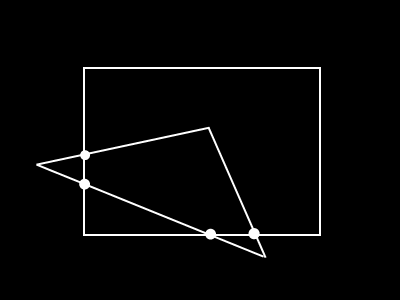
### Vertex Processing

Nhiệm vụ chính của Vertex Processing là áp dụng phép chiếu để đưa tọa độ của các đỉnh từ không gian của đối tượng về không gian màn hình. Bước này đóng vai trò quyết định đối tượng sẽ xuất hiện tại vị trí nào ở trên màn hình.



### Geometry Processing

Geometry Processing dựa vào tọa độ màn hình của mỗi đỉnh có được khi áp dụng phép chiếu ở bước Vertex Processing, để loại bỏ các thành phần không xuất hiện trong phạm vi nhìn thấy của Camera, và các bề mặt hướng ngược lại phía Camera để giảm chi phí cho GPU.

Sau đó các đỉnh và điểm trung gian nằm trong vùng nhìn thấy đều được Rasterize về tọa độ điểm ảnh (pixel) tương ứng trên màn hình.

### Pixel Processing

Pixel Processing sẽ quyết định màu sắc của đỉnh hoặc các điểm trung gian nằm trong vùng nhìn thấy.

### Pixel Rendering

Pixel Rendering là khâu cuối cùng của tiến trình vẽ một đối tượng. Tại đây, màu sắc của mỗi pixel có thể thay đổi tùy thuộc vào độ trong suốt (alpha), cách phối màu (Blending) … được thiết lập.

## Point Cloud trên Silverlight

Để có thể thu được thông tin về cảnh 3D trong thế giới thực, chúng tôi sử dụng một scanner, cụ thể là Kinect. Thông tin thu được từ Kinect bao gồm một ảnh RGB và độ sâu ứng với mỗi điểm ảnh trên ảnh RGB đó. Từ các thông tin trên, áp dụng một số kỹ thuật mà chúng tôi đã trình bày ở chương (???), chúng tôi thu được một tập hợp các điểm màu trong không gian 3D. Tập hợp các điểm màu trên được gọi là đám mây điểm (Point Cloud). Như vậy, mỗi khung cảnh sẽ tương ứng với một đám mây điểm.

Mỗi đám mây điểm bao gồm hàng trăm ngàn điểm màu. Để vẽ các đám mây điểm trên Silverlight mà vẫn đảm bảo tính ổn định về chi phí và thời gian phản hồi, chúng tôi sử dụng XNA Graphics. Nhưng tại thời điểm hiện tại, thành phần XNA Graphics được tích hợp trong Silverlight 5 Beta không hỗ trợ kiểu primitive Point.

Do đó, chúng tôi cần giả lập việc vẽ một điểm bằng cách vẽ một hình chữ nhật. Hình chữ nhật này phải đảm bảo các đặc điểm sau :

* Giữ nguyên vị trí và màu sắc của điểm.
* Luôn có kích thước tối thiểu nhưng vẫn nhận biết được trên màn hình.

Mỗi điểm P trong đám mây điểm được thể hiện thành một hình chữ nhật có tâm là P. Hình chữ nhật này bao gồm bốn đỉnh đều mang thông tin về tọa độ, màu sắc của P và vị trí tương ứng trong hình chữ nhật.

Để đảm bảo hình chữ nhật có kích thước nhỏ nhất có thể trên màn hình, ta cần can thiệp vào khâu Vertex Processing :

* Đầu tiên ta cần dùng một phép chiếu để đưa các đỉnh này từ tọa độ trong không gian về tọa độ trên màn hình bằng cách nhân tọa độ của đỉnh với một ma trận tương ứng với view của camera. (hình Vertex Processing)
* Do cả bốn đỉnh của hình chữ nhật đều có cùng chung một tọa độ nên sau phép chiếu cả bốn đỉnh đều ở cùng một vị trí trên màn hình. Khoảng cách nhỏ nhất có thể nhận biết được trên màn hình là một pixel, do đó ta lần lượt di chuyển các đỉnh này một đoạn tương ứng với một pixel để tạo thành một hình chữ nhật có kích thước tối thiểu.

## Hiệu ứng tăng cường

### Hiệu ứng chiếu sáng

Con người không trực tiếp nhìn thấy các vật thể, mà thông qua sự cảm nhận các tia sáng phản chiếu từ các vật thể đó đi đến mắt người. Tùy thuộc vào vị trí, chất liệu và góc độ của bề mặt vật thể so với nguồn sáng mà tia sáng phản chiếu đi đến mắt người có màu sắc và cường độ khác nhau.

Trong lĩnh vực đồ họa máy tính, ánh sáng được mô phỏng bằng các hiệu ứng đặc biệt còn gọi là Hiệu ứng chiếu sáng (lighting). Những hiệu ứng này giúp người xem có cảm nhận tốt hơn về chất liệu và hình dạng của vật thể, qua đó làm tính chân thật của các mô hình ba chiều.

Hiệu ứng chiếu sáng được phân ra nhiều loại nhằm tạo ra các hiệu ứng tương tự như các hiệu ứng ánh sáng tồn tại trong môi trường tự nhiên như ánh sáng mặt trời, ánh sáng từ các bóng đèn, sự phản chiếu...

Trong phạm vi đề tài, chúng tôi chỉ đề cập đến một vài hiệu ứng chiếu sáng cơ bản như Ambient, Diffusal và Specular Light.

#### Ambient Light

Thông thường, ta vẫn có thể nhìn thấy các vật thể trong một căn phòng tối bằng mắt thường. Mặc dù không có một nguồn sáng trực tiếp nào chiếu đến các vật thể này, nhưng vẫn luôn có một lượng rất nhỏ ánh sáng chiếu vào trong phòng. Ánh sáng này đến từ một nguồn sáng và được phản chiếu nhiều lần qua các vật thể khác (các nguồn sáng gián tiếp) và lọt vào trong phòng tối. Do không thể mô phỏng các tia sáng phản chiếu nhiều lần qua các vật thể như trong thể giới thật, ta dùng Ambient Light để tạo ra một hiệu ứng tương tự.

Ambient Light sẽ chiếu sáng một lượng nhỏ lên tất cả các vật thể ở tất cả các góc độ với cường độ như nhau.



#### Diffusal Light

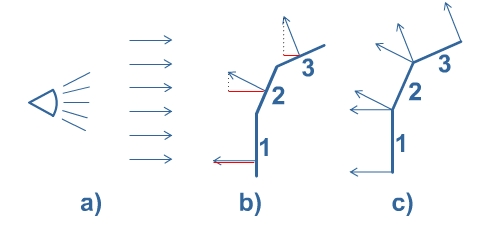
Diffusal Light là hiệu ứng mô phỏng ánh sáng phát ra từ một hoặc nhiều nguồn sáng. Ánh sáng được mô phỏng như những tia sáng (ray) do nguồn sáng phát ra và di chuyển theo đường thẳng đến các vật được chiếu sáng.

Tùy thuộc vào đặc điểm của nguồn sáng mà ta có thể phân ra thành nhiều loại. Ở đây, chúng tôi chỉ đề cập đến ba loại cơ bản là Directional Light, Spot Light và Point Light.

* Directional Light được dùng để mô phỏng sự chiếu sáng của các nguồn sáng ở xa và cường độ ánh sáng thay đổi không đáng kể trong phạm vi khung cảnh hiện tại. Các tia sáng của Directional Light đi theo các đường thẳng, song song nhau. Thông thường Directional Light được dùng để mô phỏng ánh sáng mặt trời.
* Spot Light tương tự như Directional Light nhưng ánh sáng bị giới hạn trong một hình dạng xác định. Spot Light được sử dụng để mô phỏng ánh sáng phát ra từ đèn xe hoặc đèn pin …
* Point Light được dùng để mô phỏng các nguồn sáng điểm như bóng đèn tròn, pháo sáng … Các tia sáng của Point Light đều xuất phát từ một điểm và tỏa ra tất cả các hướng xung quanh với cường độ như nhau và giảm dần khi càng cách xa nguồn sáng.



Đặc điểm chung của ánh sáng kiểu Diffusal Light là lượng ánh sáng phản chiếu từ một bề mặt vật thể còn phụ thuộc vào góc hợp bởi tia sáng và bề mặt đó.

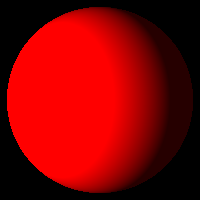
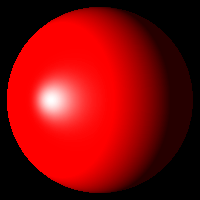


<http://www.riemers.net/eng/Tutorials/XNA/Csharp/Series1/Lighting_basics.php?expandall=1>

Độ phản chiếu của một bề mặt được tính theo công thức :

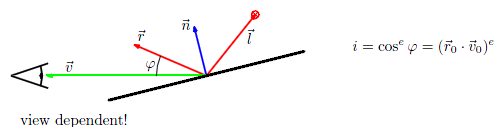
Độ phản chiếu = dot product (vector d, vector n)

#### Specular Light

<http://www.falloutsoftware.com/tutorials/gl/gl8.htm>

Khi nhìn vào một vật thể sáng bóng, ta thường thấy có một khu vực nhỏ trên vật đó sáng hơn hẳn các vùng xung quanh. Sở dĩ ta khu vực đó sáng hơn hẳn là vì các tia sáng phản xạ trên khu vực đó đi thẳng đến mắt người (phản xạ toàn phần). Điều này xảy ra khi vector vuông góc với bề mặt đó nằm ngay giữa góc hợp bởi tia sáng đến và tia nhìn (từ mắt người đến bề mặt đó). – vector v trùng với vector r



<http://www.zenofgames.com/wiki/index.php?title=Computer_Graphics_-_Basics/Lighting>

# Liên kết C++ và Silverlight

**Tóm tắt chương:**

*✍Nội dung chương trình bày các giới hạn, bảo mật nghiêm ngặt trên Silverlight, hướng giải quyết và ứng dụngđể kết nối các chương trình trên C++chạy ở Client và Silverlight dùng C#.*

## Bảo mật trên Silverlight

Sandbox, 3 tầng bảo mật (transparent, Critical, …), giới hạn tương tác với Client. Không thể vượt các tầng bảo mật (giới hạn phiên bản Silverlight 5 Beta) – chưa hỗ trợ gọi Dll.

## Tương tác với các thành phần trên Client

### Mở rộng giới hạn tương tác (Out-of-browser)

Giảm bảo mật, giới hạn tương tác được mở rộng, tương tác với các thành phần (thao tác với đường dẫn tuyệt đối của client không qua OpenFileDialog, COM component)

### COM component và các tương tác với Client.

Đọc ghi file/folder, chạy cmd, đọc registry, chạy script WMI,…

### Kích hoạt C++ từ Silverlight

Download files C++ từ ClientBin của Silverlight (từ server)

Giải nén

Kích hoạt dùng COM

### Trao đổi dữ liệu C++ và Silverlight

Silverlight dùng WMI để bắt sự kiện của 1 folder

C++ dựng thread để nhận chỉ thị thông qua tập tin cm.txt

Tài liệu tham khảo

[ ] <http://en.wikipedia.org/wiki/Kinect>

[ ] PrimeSense Inc, “NITE Controls User Guide”.

[ ] <http://www.xbox.com/en-US/Kinect/healthyfun>

[ ] <http://news.cnet.com/8301-10805_3-20035039-75.html?tag=topStories3>

[ ] <http://www.ros.org/wiki/kinect_calibration/technical>

[ ] OpenNI User Guide, *p6-10, OpenNI Library.*

[ ] PrimeSensor, “Reference Design 1.08”

[ ] Dipl. Ing. (FH) Uwe Furtner (2001), “Color processing with Bayer Mosaic sensors”, *p.1-2, Image-Processing Products Manager, MATRIX Vision GmbH.*

[ ] Rémi Jean, “Demosaicing with The Bayer Pattern”, *p.1-5, Department of Computer Science, University of North Carolina.*

[ ] Paul M. Hubel, John Liu and Rudolph J. Guttosch , “Spatial Frequency Response of Color Image Sensors: Bayer Color Filters and Foveon X3”, *p.1-2, Foveon, Inc. Santa Clara, California.*

[ ] Bradley Atcheson, “Subpixel Rendering of Bayer-Patterned Images”, *p.1-1, Department of Computer Science, The University of British Columbia.*

[ ] <https://code.ros.org/trac/opencv/ticket/1093>

[ ] K. Madsen, H.B. Nielsen, O. Tingleff (2004), “Methods for Non-Linear problems”, *p.10-11, Informatics and Mathematical Modelling, Technical University of Denmark.*

[ ] <http://en.wikipedia.org/wiki/Levenberg_Marquardt_algorithm>

[ ] <http://en.wikipedia.org/wiki/RANSAC>

[ ] Konstantinos G. Derpanis (2010), Overview of the RANSAC Algorithm.

[ ] M.A. Fischler and R.C. Bolles, “Random sample consensus: A paradigm for model ﬁtting with applications to image analysis and automated cartography”. Communications of the ACM, 24(6):381–395, 1981.