



실전

강의실 4

키넥트와 3차원 스캐너 키넥트에서 Rapidform DLL로 점군 데이터 획득하기

마이크로소프트(이하 MS)에서 출시한 키넥트(Kinect)는 Xbox 게임 컨트롤러이다. 키넥트에는 기존 게임 컨트롤러에서 흔히 볼 수 없었던 뎁스(depth) 카메라나 RGB 카메라가 내장돼 있는데, 이것을 이용하면 3차원 데이터를 활용한 프로젝트를 진행할 수 있다. 또한 키넥트를 3차원 스캐너로도 활용할 수 있다. 사실 3차원 스캐너 분야는 하드웨어가 매우 고가여서 많은 사람들이 접근하기 어려운 분야다. 하지만 키넥트에 내장된 적외선 프로젝터를 사용하면 깊이 정보를 추출해 점군(Point Cloud) 데이터를 얻을 수 있다. 이번 시간에는 키넥트 디바이스 드라이버 설치와 Rapidform DLL로 점군 데이터를 획득하는 방법을 살펴보겠다.

연재순서

- 1회 | 2011. 12 | 키넥트에서 Rapidform DLL로 점군 데이터 획득하기
- 2회 | 2012. 1 | Rapidform DLL로 획득한 데이터 최적화하기
- 3회 | 2012. 2 | 획득한 데이터를 Rapidform DLL로 측정하기



소아람 soaram@inustech.com | 서강대에서 컴퓨터그래픽으로 석사학위를 받았다. 삼성 소프트웨어 멤버십 17기이며, 프랑스 파리에서 열린 '마이크로소프트 이메진컴 2008' 임베디드 분야의 월드 파이널 리스트에 진출한 바 있다. 현재 3차원 역설계 솔루션 Rapidform을 개발하는 아이너스기술에 재직 중이다.

키넥트와 3차원 스캐너

앞서 언급한대로 MS가 선보인 키넥트는 게임 컨트롤러 용도로 제작됐다. 그럼에도 불구하고 다양한 분야에서 활용할 수 있는 하드웨어가 장착돼 있다(그림 1) 참조).

대표적인 예로 마이크가 있는데, 마이크 한 개로는 불가능하지만 마이크가 두 개 이상이면 화자의 위치를 TOF(Time of Flight) 방식으로 판별할 수 있다. 그 밖에도 가속도 센서와 제어가 가능한 모터 그리고 지금까지 살펴볼 두 개의 카메라와 적외선 프로젝터가 있다.

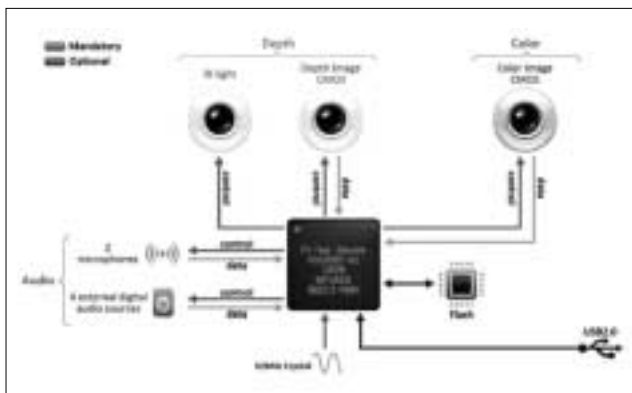


〈화면 1〉 키넥트에 내장된 카메라와 프로젝터

〈화면 1〉은 키넥트에 내장돼 있는 두 개의 카메라와 IR 프로젝터다. 키넥트에 내장된 카메라 스펙은 〈표 1〉과 같다.

구분	내용
인식 가능한 좌우 시야각	57도, 상하 43도
센서 동작이 가능 각도	± 27도
동작 깊이 인식 범위	1.2 ~ 3.5m
뎁스 카메라	320×240, 16bit, 30FPS
RGB 카메라	640×480, 32bit, 30FPS

〈표 1〉 키넥트 카메라 스펙



〈그림 1〉 키넥트 하드웨어 구조

IR 프로젝터에서 방출된 적외선은 물체에 반사된 다음 적외선 CMOS 카메라로 들어온다. 이 과정을 통해 물체의 깊이 정보를

산출할 수 있으며, 대부분의 XBox 게임이 이 정보로 관절 정보를 추정한다.

여기서 픽셀의 깊이 정보를 안다는 건 곧 3차원 스캔을 할 수 있다는 것을 뜻한다. 실제로 다양한 종류의 3차원 스캐너가 이와 유사한 방식으로 작동하는 데 대표적인 것이 광학식(optical) 스캐너다.

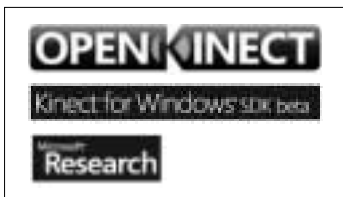
3차원 스캐너로 얻어진 점군 데이터는 3차원 좌표 정점(vertex)의 집합이다. 점군 데이터는 CAD 품질을 검사하거나 애니메이션 또는 렌더링 기법 등에 사용되며, 삼각형화(triangulation) 작업을 거치면 삼각형 매시를 만들 수 있다.

키넥트 개발 환경 설정

키넥트 발매 초기에는 공식적인 디바이스 드라이버가 없었다. 그래서 Adafruit Industries 같은 곳에서는 키넥트 드라이버를 해킹하는 이벤트성 대회를 벌이기도 했다.

그랬던 것이 키넥트 내부 알고리즘에 접근하지 않고 XBox 게임을 조작하지 않는다는 조건 하에 드라이버를 공개하기 시작했으며, 이때부터 libfreenect, CL NUI, OpenNI 등과 같은 드라이버 및 API가 공개됐다.

현재 MS 리서치에서 '키넥트 SDK 베타(Kinect for Windows SDK beta)'를 제공하고 있다. 키넥트 SDK 베타는 NUI와 함께 C#으로 개발할 수 있으며 WPF 관련 컨트롤이 툴킷으로 제공된다.



〈화면 2〉 Open Kinect, 키넥트 SDK 베타

따라서 개발 전 반드시 라이브러리를 선택해야 한다. 물론 블로그나 개인 홈페이지에 많은 예제들이 있지만 각각 다른 라이브러리를 사용하고 있어 키넥트 개발을 처음 하는 개발자들에게 혼란을 줄 수 있다. 주로 개발에 사용되는 OpenNI와 키넥트 SDK에는 개발 환경 이외에도 다소 차이가 있다.

구분	키넥트 SDK	OpenNI
운영체제	윈도우7	윈도우 XP, 비스타, 윈도우7, Mac OS X, 리눅스
언어	C++, C#	C++, C#, 자바
라이선스	비상용만 가능	상용으로 이용가능(LGPL)
지원 가능 디바이스	키넥트만 가능	키넥트, Xtion Pro

〈표 2〉 키넥트 라이브러리 차이점

〈표 2〉를 참고해 개발 목적과 환경에 맞춰 알맞은 디바이스 드라이버와 라이브러리를 사용하면 된다. 키넥트 SDK는 설치가 간편하고 제공하는 기본 예제들이 비교적 깔끔하게 구성되어 있다는 장점이 있다. 그러나 OpenNI를 사용할 경우 NITE 예제를 활용해 포즈 인식, 제스처 인식 등의 많은 예제를 활용할 수 있다.

이 글에서는 OpenKinect의 libfreenect 디바이스 드라이버를 이용해, C++로 개발한다는 것을 염두에 두길 바란다.

USB 드라이버 설치

키넥트 SDK 베타에는 USB 드라이버가 포함되어 있지만 libfreenect를 사용하려면 USB 디바이스 드라이버를 별도로 설치해야 한다. 다음 사이트에서 플랫폼에 맞는 디바이스 드라이버를 선택해 다운로드한 뒤 설치를 진행하면 된다(〈화면 3〉 참조).

- <http://openkinect.org/>



〈화면 3〉 키넥트 드라이버 설치

설치가 완료됐다면 〈화면 4〉에서 보는 것처럼 장치 관리자에서 디바이스 드라이버를 확인할 수 있다.



〈화면 4〉 장치 관리자로 드라이버 설치 확인

만약 다른 영상처리와 관련된 다양한 샘플을 실행해보고 싶다면 OpenNI 바이너리를 받아 테스트하는 것을 추천한다. 하지만 앞서 설명한 각각의 디바이스 드라이버는 서로 다른 라이브러리와 호환되지 않는다는 것을 염두에 두어야 한다.

만약 다른 라이브러리를 사용하려면 현재 설치된 디바이스 드라이버를 삭제하고 다시 설치해야 한다. 그 밖에 OpenNI 설치와 관련된 내용은 관련 홈페이지(<http://www.openni.org/downloadfiles/2-openni-binaries>)를 참조하길 바란다.

3차원 스캐닝 개발을 위한 Rapidform DLL 설치

애플리케이션 프레임워크가 될 Rapidform DLL은 다음 사이트에서 다운로드할 수 있다.

- <http://rapidform.com//dllresources>

Rapidform DLL을 설치한 다음 워크숍 예제를 실행해보자(〈화면 5〉 참조).



〈화면 5〉 Rapidform DLL의 워크숍 예제 실행

워크숍에는 Rapidform DLL에서 사용되는 다양한 API들을 간단하게 테스트할 수 있는 커맨드가 있다. 다수의 스캔 데이터를 정렬(align)하거나 점군 데이터를 삼각형화(triangulate)한 다음 메시 기반의 곡면 정보를 생성해 커브도 만들 수 있다.

또 이 커브와 평면을 이용해 메시지를 분할하거나 자를 수도 있으며, 불리언(boolean)이나 변형(deform) 같은 편집도 할 수 있다. 이밖에도 메시는 형상 편차 및 간섭 분석 등을 검사하는 기능으로도 활용할 수 있다. Rapidform DLL에 관한 다양한 기능은 관련 사이트(<http://dllsupport.rapidform.com>)를 참조하면 된다.

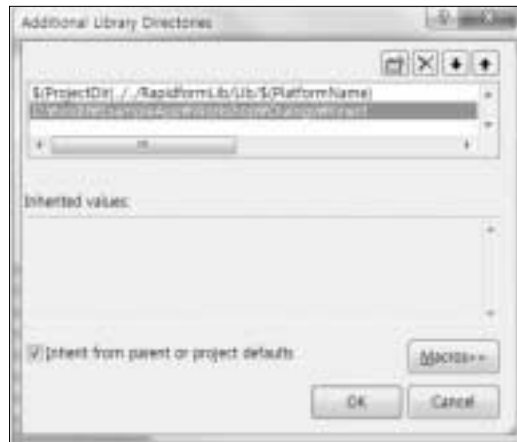
라이브러리 설치 및 프로젝트 세팅

다음 단계는 비주얼 스튜디오에서 워크숍 솔루션을 열어 키넥트 관련 파일들을 추가시키는 것이다. 추가할 파일은 〈화면 6〉과 같다.



〈화면 6〉 비주얼 스튜디오에 추가할 파일들

그 후 링크를 위해 비주얼 스튜디오의 property pages > linker > general 에서 라이브러리 파일의 디렉터리를 추가한다(〈화면 7〉 참조).



〈화면 7〉 비주얼 스튜디오에서 라이브러리 설정

키넥트 스캔을 위한 커맨드 제작

워크숍에서 동작하려면 커맨드를 먼저 제작해야 한다. 리소스 뷰에서 메뉴를 추가해 이벤트 핸들러를 등록한다. 그 다음 작업을 진행할 다이얼로그를 하나 만들어 cpp와 h를 추가한다. 추가한 라이브러리를 사용하려면 다음과 같이 정의한다.

Kinect::KinectFinder KF;

Listener *L;

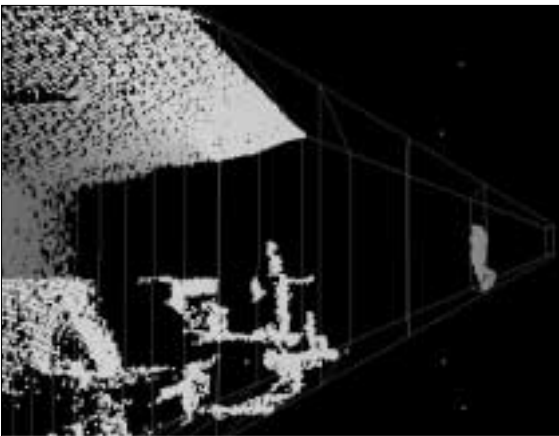
Kinect::Kinect *K;

다이얼로그에서 초기화와 관련된 함수가 〈리스트 1〉이다.

<리스트 1> 초기화 관련 함수

```
bool CDlgKinect::KinectInit()
{
    if (KF.GetKinectCount() < 1)
    {
        return false;
    }
    K = KF.GetKinect();
    if (K == 0)
    {
        return false;
    };
    L = new Listener();
    K->AddListener(L);
    printf("Kinect Init Ok.\n");
    return true;
}
```

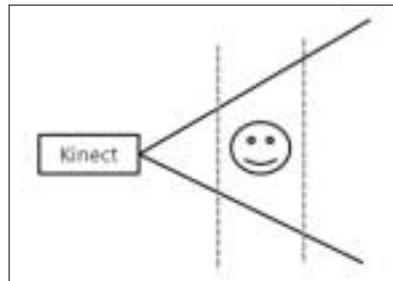
초기화를 시킨 후 점군 데이터를 획득할 영역을 설정해 LED 컨트롤을 수행시킨다. 키넥트에는 적외선 프로젝터에서 들어온 깊이와 화각 정보들이 입력되기 때문에 자연스럽게 노이즈도 들어온다. 키넥트 카메라로 인식할 수 있는 범위를 촬영한 것이 <화면 8>이다.



<화면 8> 키넥트 FOV와 깊이 정보를 RGB값으로 변환한 정보

이때 실제 스캔하려는 물체 외 정보를 삭제하려면 다음 소스 코드를 추가시켜 바운딩 박스를 정의한다.

```
RFBBox box;
box.SetMinPoint(dMinX, dMinY, dMinZ);
box.SetMaxPoint(dMaxX, dMaxY, dMaxZ);
```



<그림 2>
키넥트 FOV에서 바운딩 박스로 절투체 영역 설정

이후 점군 데이터를 추가한 부분을 다음과 같이 분기하면 더욱 깔끔한 데이터를 얻을 수 있다. 이 소스 코드에서 mVertices 클래스는 3차원 위치 정보와 색상 정보 등을 담고 있다.

```
if (box.IsPointIn(mVertices[id].x, mVertices[id].y, mVertices[id].z))
```

다음은 모터를 제어하는 소스 코드이다.

```
K->SetMotorPosition(dMortor);
```

LED를 제어하려면 다음 소스 코드를 더한다.

```
K->SetLedMode(Kinect::Led_Yellow);
```

바운딩박스 크기와 LED 색상과 모터를 컨트롤하는 다이얼로그는 <화면 9>와 같이 구성돼 있다.



<화면 9>
키넥트 컨트롤을 위한 다이얼로그

지금부터는 스캔 버튼을 눌렀을 때 수행될 소스 코드를 작성해 보자.

〈리스트 2〉 스캔 버튼을 눌러 수행시키는 소스 코드

```

mDepthFrameCounter++;
K->ParseDepthBuffer();
int i = 0;
for (int y = 0; y < 480; y++)
{
    unsigned char *destrow = DepthColor + ((479-
y)*(640))*3;
    float *destdepth = mDepthBuffer + ((479-
y)*(640));
    float *maxdestdepth = mMaxDepthBuffer +
((479-y)*(640));
    for (int x = 0; x < 640; x++)
    {
        unsigned short Depth = K-
>mDepthBuffer[i];
        if (Depth > 0 && Depth != 0x07ff)
        {
            float D = 100.0f / (-0.00307f
* Depth + 3.33f);
            *destdepth++ = D;
            if (D > *maxdestdepth )
            {
                *maxdestdepth = D-40;
            }
            maxdestdepth++;
        }
        else
        {
            *destdepth++ = -100000;
            maxdestdepth++ ;
        }
        i++;
    }
};

if (mDepthFrameCounter != mLastDepthFrameCounter)
{
    for (int y = 0; y < 480; y += BLOBDIV)
    {
        for (int x = 0; x < 640; x += BLOBDIV)
        {
            int yy = y / BLOBDIV;
            int xx = x / BLOBDIV;
            mBFrame[yy][xx] =
mDepthBuffer[(x+y*640)];
        }
    }
    memcpy(mOldBFrame, mBFrame,
(640 / BLOBDIV) * (480 / BLOBDIV) * sizeof(float));
}
    
```

```

mLastDepthFrameCounter = mDepthFrameCounter;

for (int y = 0; y < 480; y += 3)
{
    for (int x = 0; x < 640; x += 3)
    {
        int idx = (x+y*640);
        float zz = mDepthBuffer[idx];
        bool skip = false;

        if (zz > -90000 && !skip)
        {
            mVertices[id].x = (float)x;
            mVertices[id].y = (float)y;
            mVertices[id].z = zz;

            Kinect::KinectDepthToWorld(mVertices[id].x, mVertices[id].y
, mVertices[id].z);

            mVertices[id].u =
mVertices[id].x;
            mVertices[id].v =
mVertices[id].y;

            Kinect::KinectWorldToRGBSpace(mVertices[id].u, mVertices[id
].v, mVertices[id].z);

            int idx =
((int)(mVertices[id].u) + (479 - (int)mVertices[id].v) * 640) * 3;

            mVertices[id].r = K-
>mColorBuffer[idx+0];
            mVertices[id].g = K-
>mColorBuffer[idx+1];
            mVertices[id].b = K-
>mColorBuffer[idx+2];

            if (skip)
            {
                mVertices[id].r
= 255;
                mVertices[id].g = 0;
                mVertices[id].b = 0;
            }
            id++;
        }
    }
};
    
```

〈리스트 2〉에서 mVertices에는 픽셀에서 깊이 정보를 이용해 Z값을 추정한다. 이제 X, Y, Z값을 모두 구했으니 그리기만 하면 된다. 위크숍 안에 있는 점군 데이터 타입을 추가하는 Rapidform DLL API를 〈리스트 3〉과 같이 호출해보자.

<리스트 3> 점군 데이터 타입 추가 API

```

RFEntMeshID eidMesh;
    rfModel::AddMesh(_T("Kinect Scan Data"), eidMesh,
RF_FLOAT_VERTEX);
    RFEntVertexID eidVtDummy;
    rfMesh::SetMaterialType(eidMesh,
RF_COLOR_PER_VERTEX);

    for (int i = 0; i < 480; i++)
    {
        for (int j=0; j<640; j++)
        {
            int idx =
((int)(mVertices[id].u)+(479-(int)mVertices[id].v)*640)*3;

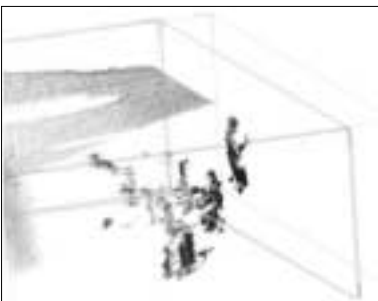
            mVertices[id].r = K-
>mColorBuffer[idx+0];
            mVertices[id].g = K-
>mColorBuffer[idx+1];
            mVertices[id].b = K-
>mColorBuffer[idx+2];

            rfMesh::AddVertex(eidMesh,
mVertices[id].x, mVertices[id].y, mVertices[id].z,
mVertices[id].r, mVertices[id].g, mVertices[id].b, 255,
eidVtDummy);

                id++;
        }
    }

    rfMesh::Regenerate(eidMesh, false);
    
```

결과화면



<화면 10>
<그림 2>의 절두체를 설정하지 않은 경우

지금까지 만든 소스 코드를 종합하면 <화면 10>과 <화면 11>에서 보는 것과 같은 점군 데이터를 얻을 수 있다. 두 화면의 다른 점은 <화면 10>은 절두체를 설정하지 않아 얻으려는 물체 외 정보가 있는 반면 <화면 11>은 절두체를 설정해 물체 정보만을 얻어낸 것이다.



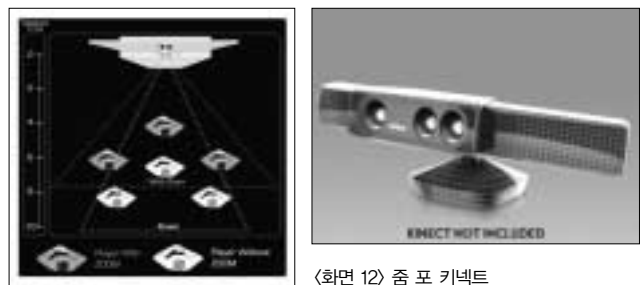
<화면 11> <그림 2>의 절두체를 설정한 경우

정리하며

이번 시간에는 키넥트의 설치 방법부터 점군 데이터를 획득하는 것까지 알아봤다.

좀 더 세밀한 3차원 데이터를 얻으려면 카메라의 화각과 최소 거리를 고려해야 한다. 키넥트 카메라는 기본적으로 50~80cm의 최소 거리가 필요하다. 따라서 크기가 작은 물체에 화각을 맞춰 스캔하는 것은 쉽지 않다.

이를 보완하기 위해 Nyko에서는 '줌 포 키넥트(Zoom for Kinect)'라는 액세서리를 선보인 바 있다. 이 액세서리를 <화면 12>와 같이 렌즈에 장착하면 최소 인식거리를 40% 정도 줄일 수 있다.



<화면 12> 줌 포 키넥트

이번 시간에 살펴본 내용만을 가지고는 키넥트가 완전한 3차원 스캐너라고 말할 수 없다. 왜냐하면 정렬(alignment)과 정합(registration)으로 완전한 스캔을 하려면 여러 장의 이미지를 사용해야 하기 때문이다. 다음 시간에는 여러 장의 이미지를 정합하는 방법과 Rapidform DLL에서 사용되는 정렬 방식으로 키넥트 점군 데이터를 정렬하는 법을 살펴보고자다. ●