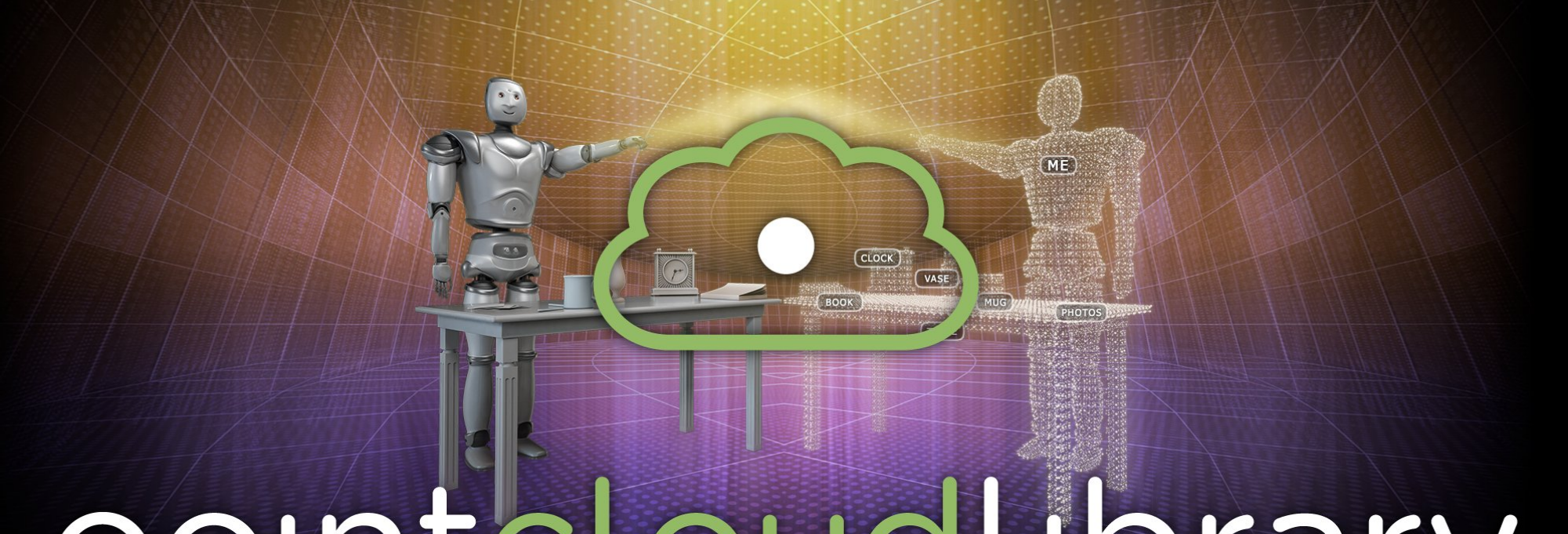



pointcloudlibrary

# PCL :: I/O

**Suat Gedikli, Nico Blodow**

July 1, 2011

pointcloudlibrary

# Outline

1. Reading and Writing PCD Files
2. The PCD File Format
3. The Grabber Interface
4. Writing a Custom Grabber

pointcloudlibrary

# Reading PCD Files 1

global functions in the namespace pcl::io

## Reading ROS-type point clouds - simple version

```
inline int loadPCDFile (const std::string &file_name, sensor_msgs::PointCloud2 &
cloud);
```

## extended version reading the sensor pose

```
inline int loadPCDFile (const std::string &file_name, sensor_msgs::PointCloud2 &
cloud, Eigen::Vector4f &origin, Eigen::Quaternionf &orientation);
```

## Reading native point clouds

```
template<typename PointT> inline int loadPCDFile (const std::string &file_name,
pcl::PointCloud<PointT> &cloud);
```

pointcloudlibrary

# Reading PCD Files 2

The pcl::FileReader interface - e.g. pcl::PCDReader

## Reading the header (meta data) of a PCD file.

```
int readHeader (const std::string &file_name, sensor_msgs::PointCloud2 &cloud,
Eigen::Vector4f &origin, Eigen::Quaternionf &orientation, int &file_version, bool
 &binary_data, int &data_idx);
```

## Reading ROS-type point clouds - simple version

```
int read (const std::string &file_name, sensor_msgs::PointCloud2 &cloud);
```

## extended version reading the sensor pose

```
int read (const std::string &file_name, sensor_msgs::PointCloud2 &cloud, Eigen::
Vector4f &origin, Eigen::Quaternionf &orientation, int &file_version);
```

## Reading native point clouds

```
template<typename PointT> int read (const string &file_name, PointCloud<PointT> &
cloud);
```

pointcloudlibrary

# Writing PCD Files 1

global functions in the namespace pcl::io

## writing ROS-type point clouds

```
int savePCDFile (string file_name, const PointCloud2 &cloud, const Vector4f &
origin, const Quaternionf &orientation, bool binary_mode);
```

## writing native point clouds

```
template<typename PointT> int savePCDFile (const string &file_name, const
PointCloud<PointT> &cloud, bool binary_mode);
```

```
template<typename PointT> int savePCDFile (const std::string &file_name, const
pcl::PointCloud<PointT> &cloud, const std::vector<int> &indices, bool binary_mode
 = false);
```

pointcloudlibrary

# Writing PCD Files 2

The pcl::FileWriter interface - e.g. pcl::PCDWriter

## writing ROS-type point clouds

```
inline int write (const std::string &file_name, const sensor_msgs::PointCloud2
&cloud, const Eigen::Vector4f &origin = Eigen::Vector4f::Zero (), const Eigen::
Quaternionf &orientation = Eigen::Quaternionf::Identity (), bool binary = false);
```

## writing native point clouds

```
template<typename PointT> inline int write (const std::string &file_name, const
pcl::PointCloud<PointT> &cloud, bool binary = false);
```

pointcloudlibrary

# The PCD File Format

## ASCII

examples/pcd_write.cpp assembles an ordered point cloud and writes it as an ASCII pcd file.

```
1   #include <pcl/io/pcd_io.h>
2   #include <pcl/point_types.h>
3
4   int main ()
5   {
6     pcl::PointCloud<pcl::PointXYZRGB> cloud;
7     cloud.points.resize (10);
8     cloud.height = 2;
9     cloud.width = 5;
10    cloud.sensor_origin_[0] = 11.1;
11    cloud.sensor_origin_[1] = 22.2;
12    cloud.sensor_origin_[2] = 33.3;
13    cloud.sensor_orientation_.w () = 0.1;
14    cloud.sensor_orientation_.x () = 0.2;
15    cloud.sensor_orientation_.y () = 0.3;
16    cloud.sensor_orientation_.z () = 0.4;
17
18    for (unsigned point_idx = 0; point_idx < cloud.points.size(); ++point_idx)
19    {
20      cloud.points[point_idx].x = 1.1 + point_idx;
21      cloud.points[point_idx].y = 1.2 + point_idx;
22      cloud.points[point_idx].z = 1.3 + point_idx;
23      cloud.points[point_idx].rgb = 0.0;
24    }
25
26    return pcl::io::savePCDFile ("test.pcd", cloud, false);
27  }
```

pointcloudlibrary

# The PCD File Format

## ASCII output

### the output PCD file

```
1   # .PCD v.7 - Point Cloud Data file format
2   VERSION .7
3   FIELDS x y z rgb
4   SIZE 4 4 4 4
5   TYPE F F F F
6   COUNT 1 1 1 1
7   WIDTH 5
8   HEIGHT 2
9   VIEWPOINT 11.1 22.2 33.3 0.1 0.2 0.3 0.4
10  POINTS 10
11  DATA ascii
12  1.1 1.2 1.3 0
13  2.1 2.2 2.3 0
14  3.1 3.2 3.3 0
15  4.1 4.2 4.3 0
16  5.1 5.2 5.3 0
17  6.1 6.2 6.3 0
18  7.1 7.2 7.3 0
19  8.1 8.2 8.3 0
20  9.1 9.2 9.3 0
21  10.1 10.2 10.3 0
```

pointcloudlibrary

# The PCD File Format

## Binary output

### the output PCD file

```
1   # .PCD v.7 - Point Cloud Data file format
2   VERSION .7
3   FIELDS x y z _ rgb _
4   SIZE 4 4 4 1 4 1
5   TYPE F F F U F U
6   COUNT 1 1 1 4 1 12
7   WIDTH 5
8   HEIGHT 2
9   VIEWPOINT 11.1 22.2 33.3 0.1 0.2 0.3 0.4
10  POINTS 10
11  DATA binary
12  <Binary Data>
```

pointcloudlibrary

# PCD: ASCII vs. Binary

Advantages of the ASCII PCD format

- human readable $\rightarrow$ can be edited manually
- can be exchanged through all machine architectures

pointcloudlibrary

# PCD: ASCII vs. Binary

Advantages of the ASCII PCD format

- human readable $\rightarrow$ can be edited manually
- can be exchanged through all machine architectures

Advantages of the Binary PCD format

- very fast, since one mmap call
- usually smaller

pointcloudlibrary

# The Grabber Interface

- polymorphic - easily exchangeable (e.g. OpenNIGrabber, PCDGrabber, ONIGrabber)
- extensible - allows all kind of data to be published
- supports streaming devices (OpenNIGrabber) as well as triggered devices (PCDGrabber)

pointcloudlibrary

# The Grabber Interface

- polymorphic - easily exchangeable (e.g. OpenNIGrabber, PCDGrabber, ONIGrabber)
- extensible - allows all kind of data to be published
- supports streaming devices (OpenNIGrabber) as well as triggered devices (PCDGrabber)

```
template<typename T> boost::signals2::connection registerCallback (const boost::
function<T>& callback)throw (pcl::PCLIOException);
```

```
template<typename T> bool providesCallback ()const;
```

```
virtual void start ()throw (pcl::PCLIOException)= 0 ;
```

```
virtual void stop ()throw (pcl::PCLIOException)= 0 ;
```

```
virtual std::string getName ()const = 0;
```

```
virtual bool isRunning ()const throw (pcl::PCLIOException)= 0;
```

pointcloudlibrary

# The Grabber Interface

- polymorphic - easily exchangeable (e.g. OpenNIGrabber, PCDGrabber, ONIGrabber)
- extensible - allows all kind of data to be published
- supports streaming devices (OpenNIGrabber) as well as triggered devices (PCDGrabber)

```
template<typename T> boost::signals2::connection registerCallback (const boost::
function<T>& callback)throw (pcl::PCLIOException);
```

```
template<typename T> bool providesCallback ()const;
```

```
virtual void start ()throw (pcl::PCLIOException)= 0 ;
```

```
virtual void stop ()throw (pcl::PCLIOException)= 0 ;
```

```
virtual std::string getName ()const = 0;
```

```
virtual bool isRunning ()const throw (pcl::PCLIOException)= 0;
```

Note:

For triggered devices, start is used to trigger, stop has no effect and isRunning is always false.

For streaming devices, start has to be implemented non-blocking!

pointcloudlibrary

# Using the Grabber Interface

## Using Global Callback Functions

Let's have a look at the example application examples/openni_grabber_example.cpp.

create a new grabber object of type pcl::OpenNIGrabber

```
65 pcl::Grabber* grabber = new pcl::OpenNIGrabber();
```

pointcloudlibrary

# Using the Grabber Interface

## Using Global Callback Functions

Let's have a look at the example application examples/openni_grabber_example.cpp.

create a new grabber object of type pcl::OpenNIGrabber

```
65 pcl::Grabber* grabber = new pcl::OpenNIGrabber();
```

check wheter the grabber provides a point cloud of type PointCloud<pcl::PointXYZRGB>

```
72   if (grabber->providesCallback<void (const pcl::PointCloud<pcl::PointXYZRGB>::
ConstPtr&)>())
```

pointcloudlibrary

# Using the Grabber Interface

## Using Global Callback Functions

Let's have a look at the example application examples/openni_grabber_example.cpp.

create a new grabber object of type pcl::OpenNIGrabber

```
65 pcl::Grabber* grabber = new pcl::OpenNIGrabber();
```

check wheter the grabber provides a point cloud of type PointCloud<pcl::PointXYZRGB>

```
72   if (grabber->providesCallback<void (const pcl::PointCloud<pcl::PointXYZRGB>::
ConstPtr&)>())
```

create a boost::function object using boost::bind

```
74     boost::function<void (const pcl::PointCloud<pcl::PointXYZRGB>::ConstPtr&)>
global_function = boost::bind (globalFunction, _1);
```

pointcloudlibrary

# Using the Grabber Interface

## Using Global Callback Functions

Let's have a look at the example application examples/openni_grabber_example.cpp.

create a new grabber object of type pcl::OpenNIGrabber

```
65 pcl::Grabber* grabber = new pcl::OpenNIGrabber();
```

check wheter the grabber provides a point cloud of type PointCloud<pcl::PointXYZRGB>

```
72   if (grabber->providesCallback<void (const pcl::PointCloud<pcl::PointXYZRGB>::
ConstPtr&)>())
```

create a boost::function object using boost::bind

```
74     boost::function<void (const pcl::PointCloud<pcl::PointXYZRGB>::ConstPtr&)>
global_function = boost::bind (globalFunction, _1);
```

register the function object

```
76 grabber->registerCallback(global_function);
```

pointcloudlibrary

# Using the Grabber Interface

## Using Global Callback Functions

Let's have a look at the example application examples/openni_grabber_example.cpp.

create a new grabber object of type pcl::OpenNIGrabber

```
65 pcl::Grabber* grabber = new pcl::OpenNIGrabber();
```

check wheter the grabber provides a point cloud of type PointCloud<pcl::PointXYZRGB>

```
72   if (grabber->providesCallback<void (const pcl::PointCloud<pcl::PointXYZRGB>::
ConstPtr&)>())
```

create a boost::function object using boost::bind

```
74     boost::function<void (const pcl::PointCloud<pcl::PointXYZRGB>::ConstPtr&)>
global_function = boost::bind (globalFunction, _1);
```

register the function object

```
76 grabber->registerCallback(global_function);
```

start the grabber

```
79 grabber->start();
```

pointcloudlibrary

# Using the Grabber Interface

## Using Member Callback Functions

Let's have a look at the example application examples/openni_grabber_example.cpp.

create a new grabber object of type pcl::OpenNIGrabber

```
65 pcl::Grabber* grabber = new pcl::OpenNIGrabber();
```

create an instance of the point cloud processor

```
66 SimpleOpenNIProcessor processor;
```

pointcloudlibrary

# Using the Grabber Interface

## Using Member Callback Functions

Let's have a look at the example application examples/openni_grabber_example.cpp.

create a new grabber object of type pcl::OpenNIGrabber

```
65 pcl::Grabber* grabber = new pcl::OpenNIGrabber();
```

create an instance of the point cloud processor

```
66 SimpleOpenNIProcessor processor;
```

create a boost::function object using boost::bind

```
68       boost::function<void (const pcl::PointCloud<pcl::PointXYZ>::ConstPtr&)>
member_function = boost::bind (&SimpleOpenNIProcessor::memberFunction, &processor
, _1);
```

pointcloudlibrary

# Using the Grabber Interface

## Using Member Callback Functions

Let's have a look at the example application examples/openni_grabber_example.cpp.

create a new grabber object of type pcl::OpenNIGrabber

```
65 pcl::Grabber* grabber = new pcl::OpenNIGrabber();
```

create an instance of the point cloud processor

```
66 SimpleOpenNIProcessor processor;
```

create a boost::function object using boost::bind

```
68       boost::function<void (const pcl::PointCloud<pcl::PointXYZ>::ConstPtr&)>
member_function = boost::bind (&SimpleOpenNIProcessor::memberFunction, &processor
, _1);
```

register the function object

```
70 grabber->registerCallback(member_function);
```

pointcloudlibrary

# Using the Grabber Interface

## Using Member Callback Functions

Let's have a look at the example application examples/openni_grabber_example.cpp.

create a new grabber object of type pcl::OpenNIGrabber

```
65 pcl::Grabber* grabber = new pcl::OpenNIGrabber();
```

create an instance of the point cloud processor

```
66 SimpleOpenNIProcessor processor;
```

create a boost::function object using boost::bind

```
68        boost::function<void (const pcl::PointCloud<pcl::PointXYZ>::ConstPtr&)>
member_function = boost::bind (&SimpleOpenNIProcessor::memberFunction, &processor
, _1);
```

register the function object

```
70 grabber->registerCallback(member_function);
```

start the grabber

```
79 grabber->start();
```

pointcloudlibrary

# Writing a Grabber

examples/sphere_grabber.h

```
33    boost::signals2::signal<void (const boost::shared_ptr<const pcl::PointCloud<
pcl::PointXYZRGB> >&)>* point_cloud_rgb_signal_;
```

pointcloudlibrary

# Writing a Grabber

examples/sphere_grabber.h

```
33   boost::signals2::signal<void (const boost::shared_ptr<const pcl::PointCloud<
pcl::PointXYZRGB> >&)>* point_cloud_rgb_signal_;
```

examples/sphere_grabber.cpp

create the signal object and store a pointer to it

```
29   point_cloud_rgb_signal_ = createSignal <void (const boost::shared_ptr<const
pcl::PointCloud<pcl::PointXYZRGB> >&)>();
```

create the point cloud

```
52           boost::shared_ptr<const pcl::PointCloud<pcl::PointXYZRGB> > cloud =
assembleSphere ();
```

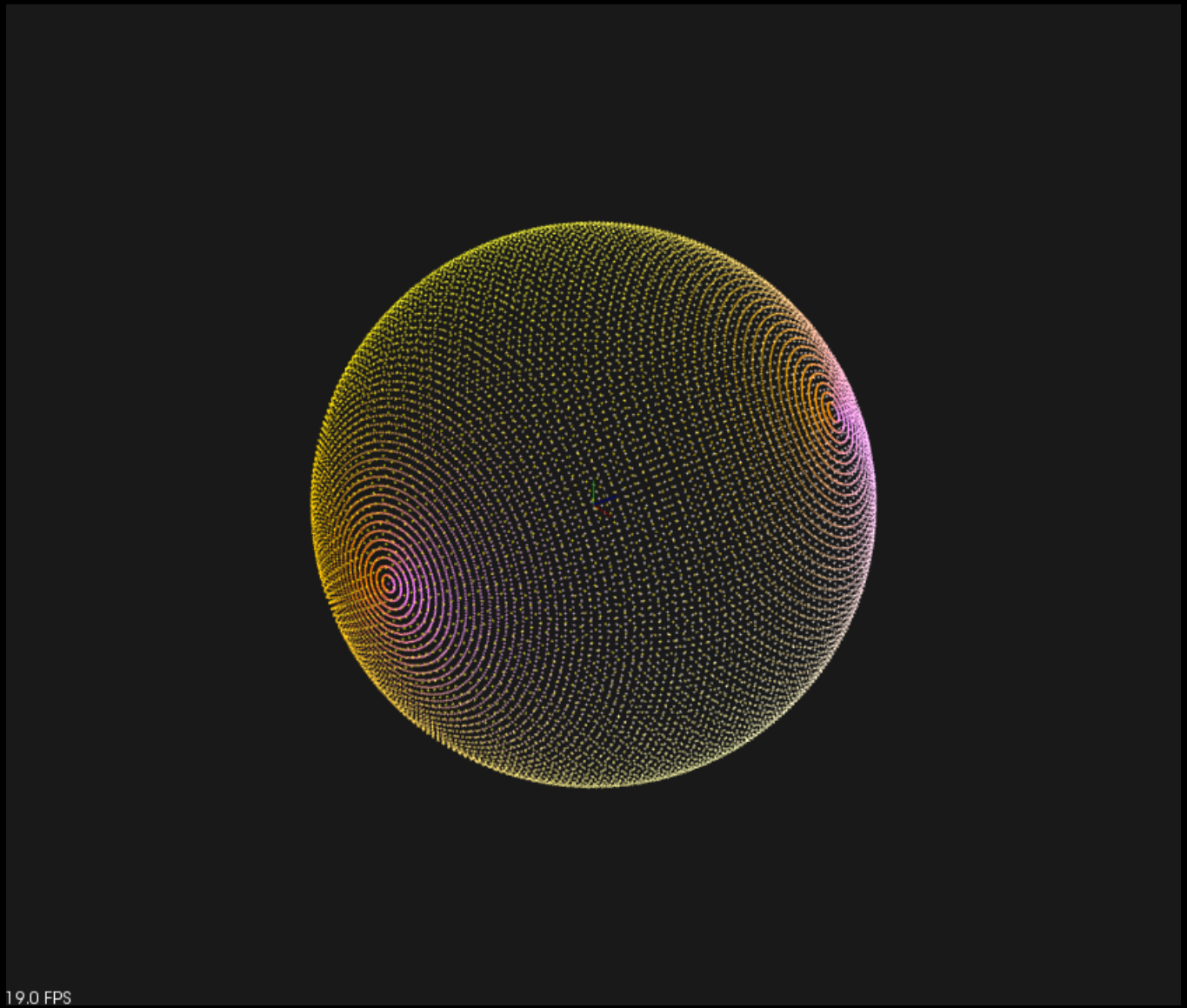
publish the point cloud

```
55 point_cloud_rgb_signal_->operator()(cloud);
```

or

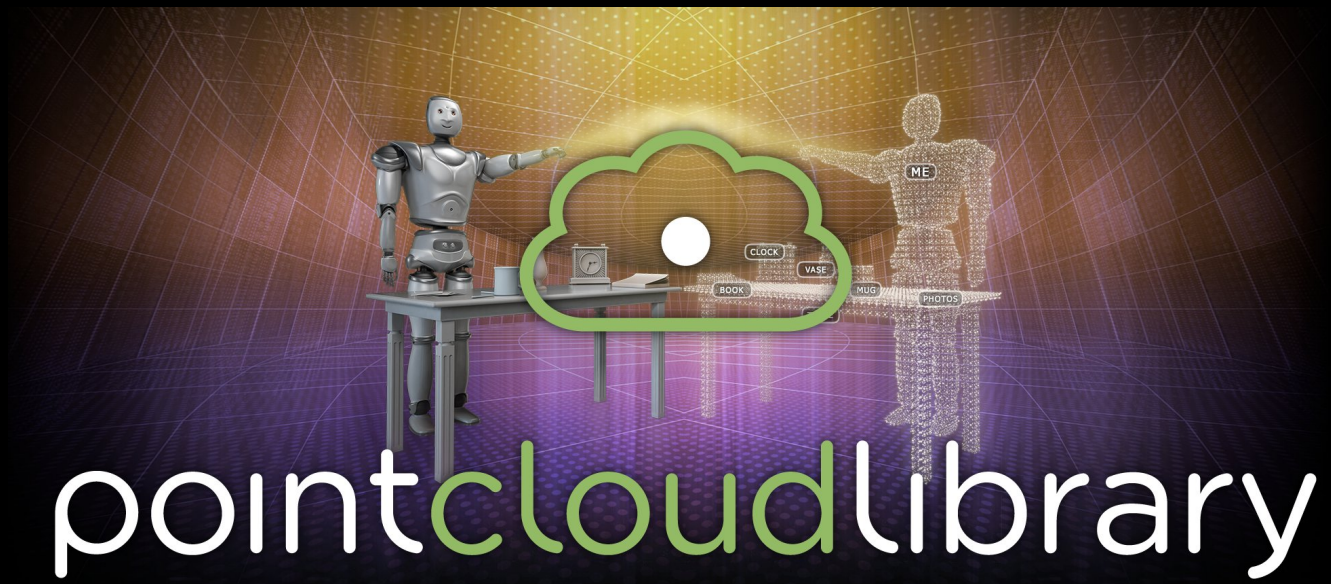
```
56 (*point_cloud_rgb_signal_)(cloud);
```

# examples/sphere_grabber_example.cpp



pointcloudlibrary

# Thanks!




http://pointclouds.org/