



Universidad Católica
San Pablo



CONCYTEC

Unique Signatures of Histograms for Local Surface Description

ESTUDIANTE DE MAESTRÍA
ALEJANDRA CALLO - JOSE JAITA

¿Qué es un descriptor?

Un descriptor extrae información geométrica de un objeto 3D, es decir una nube de puntos

Shot

Es un algoritmo híbrido que combina los histogramas con las firmas para la mejor extracción de datos, accediendo al color y su forma tridimensional,

Recolecta información de varios histogramas alrededor imitando el comportamiento de los métodos por firma. Para la firma se usa una malla esférica Almacenando los valores radiales y la elevaciones en los ejes.

Abre un archivo . ply



Extrae su descriptor



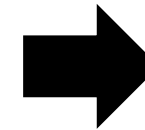
Modifica la malla de
puntos(rotacion y
ruido)



Extrae un nuevo
descriptor



Comparación



Invarianza SHOT

Compiler

Requerimientos:

- VTK
- CMAKE

Modifica

```
vtkPolyData * pd = vtkPolyData :: New ( ) ;  
someAlgorithm - > SetInputConnection ( pd - > GetProducerPort ( ) ) ;
```

Esto puede ser reemplazado por:

```
vtkPolyData * pd = vtkPolyData :: New ( ) ;  
someAlgorithm - > SetInputData ( pd ) ;
```

Entrada

La entrada es un archivo en formato .PLY que sirve para almacenar objetos gráficos que se describen como una colección de polígonos. El formato de archivo tiene dos subformatos: una representación ASCII y una versión binaria.



Función de covarianza

Según los trabajos de Hoppe y Mitra se presenta una matriz de covarianza basada en la descomposición de los autovalores, de los k-vecinos mas cercanos al punto P_i

$$M = \frac{1}{k} \sum_{i=0}^k (P_i - \hat{P})(P_i - \hat{P})^T, \quad \hat{P} = \frac{1}{k} \sum_{i=1}^k P_i .$$

Se modifica para que a los puntos mas lejanos tengan pesos pequeños, a fin de aumentar la repetibilidad para mejorar la solidez al ruido manteniendo el Radio (R) que se utilizan para calcular el descriptor. Reemplazando el centro con el punto de característica P

$$M = \frac{1}{\sum_{i: d_i \leq R} (R - d_i)} \sum_{i: d_i \leq R} (R - d_i) (P_i - P)(P_i - P)^T$$

```
void getSHOTLocalRF(vtkPolyData *cloud, vtkIdList *NNpoints, double radius, int index, float *rfc)
{
    double originDouble[3];
    cloud->GetPoint(index, originDouble);

    int nNeighbours = NNpoints->GetNumberOfIds();

    //double V_Vt[9];
    double* currPoint;
    double* vij = new double[nNeighbours * 3];
    double *covM[3];

    // Initialize covariance matrix
    covM[0] = new double[3];
    covM[1] = new double[3];
    covM[2] = new double[3];
    memset(covM[0], 0.0, sizeof(double)*3);
    memset(covM[1], 0.0, sizeof(double)*3);
    memset(covM[2], 0.0, sizeof(double)*3);

    memset(covM[0], 0, sizeof(double)*3);
    memset(covM[1], 0, sizeof(double)*3);
    memset(covM[2], 0, sizeof(double)*3);

    double distance = 0.0;
    double sum = 0.0;

    int validNNpoints = 0;
```

Inicializa la matriz


```

for(int ne = 0; ne < nNeighbours; ne++)
{

    if(NNpoints->GetId(ne) != index) { // perchè il KdTree restituisce anche il punto origine

        currPoint = cloud->GetPoint(NNpoints->GetId(ne));

        // Difference between current point and origin
        vij[validNNpoints*3 + 0] = currPoint[0] - originDouble[0];
        vij[validNNpoints*3 + 1] = currPoint[1] - originDouble[1];
        vij[validNNpoints*3 + 2] = currPoint[2] - originDouble[2];

        distance = radius - sqrt(vij[validNNpoints*3 + 0]*vij[validNNpoints*3 + 0]
        + vij[validNNpoints*3 + 1]*vij[validNNpoints*3 + 1] + vij[validNNpoints*3 + 2]*vij[validNNpoints*3 + 2]);

        // Multiply vij * vij'
        covM[0][0] += distance * vij[validNNpoints*3] * vij[validNNpoints*3];
        covM[1][1] += distance * vij[validNNpoints*3+1] * vij[validNNpoints*3+1];
        covM[2][2] += distance * vij[validNNpoints*3+2] * vij[validNNpoints*3+2];

        double temp = distance * vij[validNNpoints*3] * vij[validNNpoints*3+1];
        covM[0][1] += temp;
        covM[1][0] += temp;
        temp = distance * vij[validNNpoints*3] * vij[validNNpoints*3+2];
        covM[0][2] += temp;
        covM[2][0] += temp;
        temp = distance * vij[validNNpoints*3+1] * vij[validNNpoints*3+2];
        covM[1][2] += temp;
        covM[2][1] += temp;

        sum += distance;
        validNNpoints++;

    }
}

```

Diferencia entre el punto actual y origen

Distancia = Radio – Distancia Euclidea

Multiplica Vij * Vij'

```

covM[0][0] /= sum; covM[0][1] /= sum;
covM[0][2] /= sum; covM[1][0] /= sum;
covM[1][1] /= sum; covM[1][2] /= sum;

covM[2][0] /= sum; covM[2][1] /= sum;
covM[2][2] /= sum;

```

```

double eval[3];
double *evect[3];
evect[0] = new double[3];
evect[1] = new double[3];
evect[2] = new double[3];

```

```

// Diagonalization (eval = eigenvalues, evect = eigenvector)
// - Eigenvalues and eigenvectors are sorted in decreasing order
// - Eigenvectors are already normalized
int resJ = vtkMath::Jacobi(covM, eval, evect);

```

```

int plusNormal = 0, plusTangentDirection1=0;

```

```

for(int ne = 0; ne < validNNpoints; ne++)
{
    double dotProduct = vij[ne*3]*evect[0][0] + vij[ne*3 + 1]*evect[1][0] + vij[ne*3 + 2]*evect[2][0];
    if (dotProduct >= 0)
    {
        plusTangentDirection1++;
    }
    dotProduct = vij[ne*3]*evect[0][2] + vij[ne*3 + 1]*evect[1][2] + vij[ne*3 + 2]*evect[2][2];
    if (dotProduct >= 0)
    {
        plusNormal++;
    }
}

```

Se halla Autovalores y Autovectores

Ordena y Normaliza

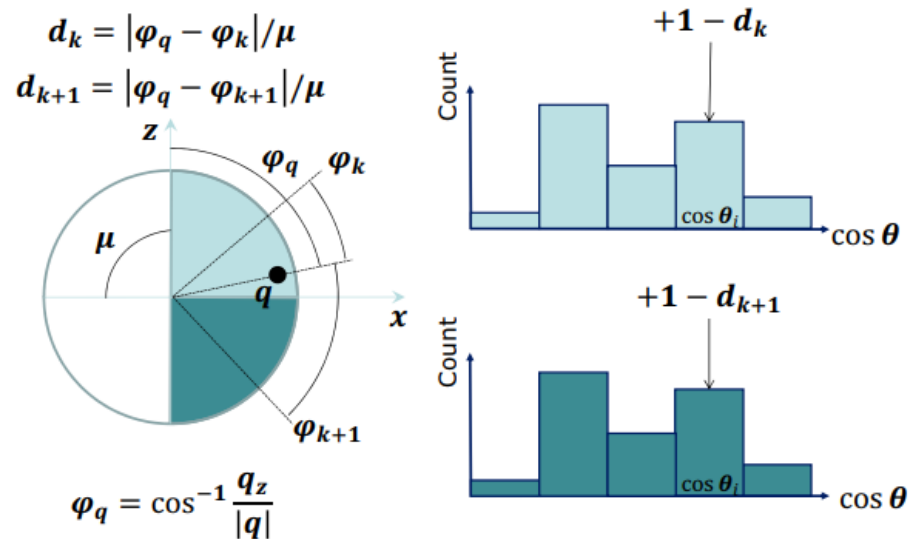
Se halla Tangentes

Producto Punto

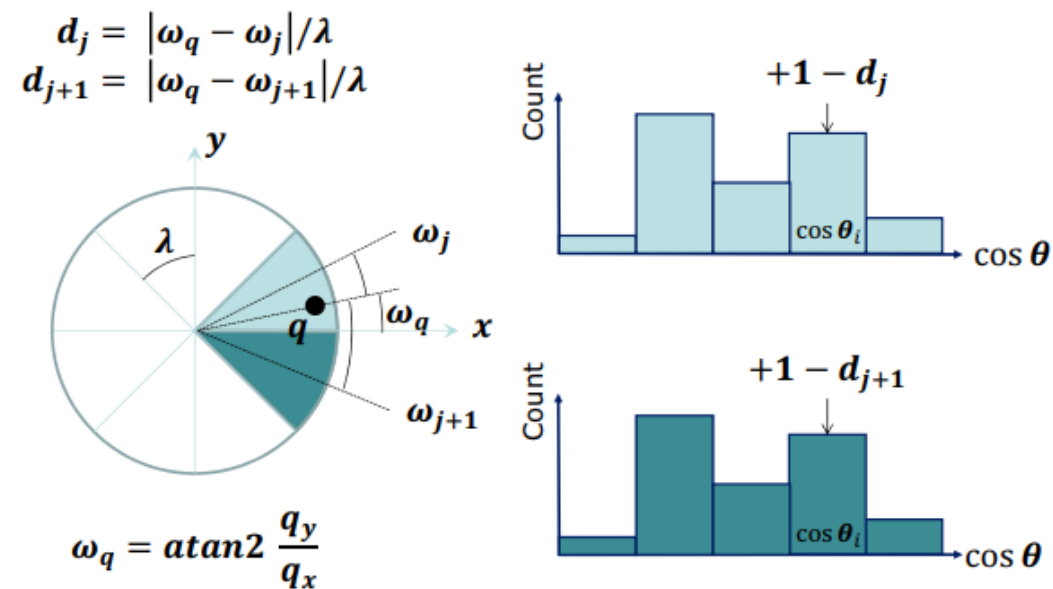
Si (color)

RGB->CIELAB

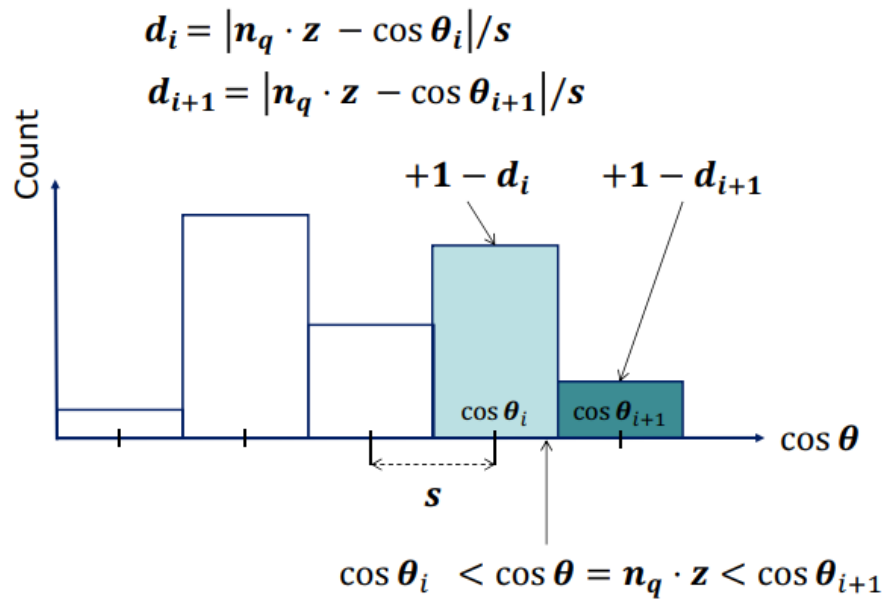
Interpolación de Canales



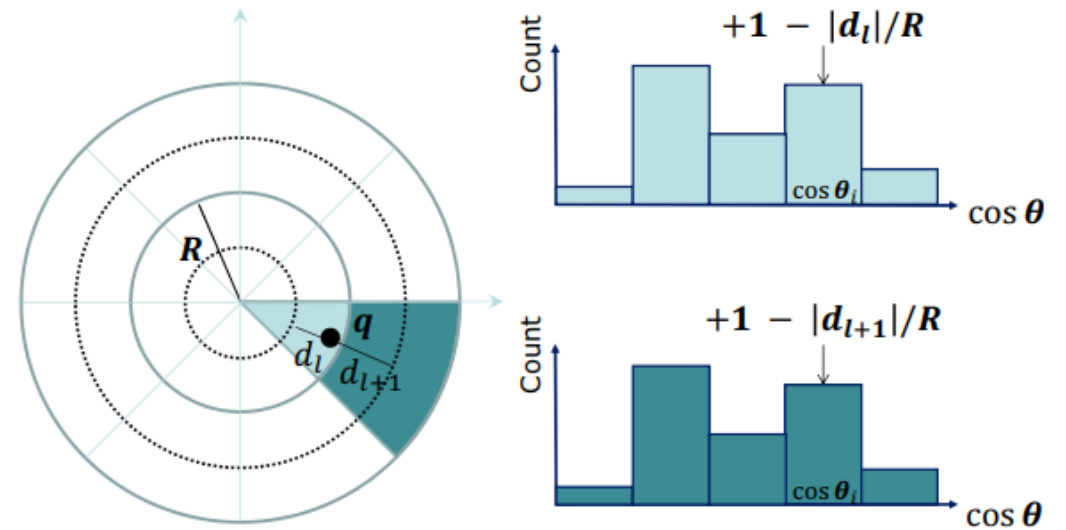
(c) Interpolation on elevation



(b) Interpolation on azimuth



(a) Interpolation on normal cosines



(d) Interpolation on distance

La Función cuadrilineal logra robustez a las variaciones de densidad, normaliza el conjunto descriptor para tener la norma euclidiana igual a 1.

```
void SHOTDescriptor::interpolateDoubleChannel
```

Función de rotación

Ruido Gaussiano

Calcula la normales

Estructura KdTree-> información tridimensional

32 volúmenes

8 divisiones azimuth

2 divisiones elevación

2 divisiones radiales

Longitud es 352

De la Ejecución del Código y la Salida

- Después de abrir el archivo .ply se extrae el descriptor.
- Se modifica esta malla de puntos en su rotación o el ruido
- Se vuelve a extraer el descriptor y se compara con la entrada
- Se prueba que el método SHOT es invariante y robusto a la rotación y el ruido haciendo un match entre el descriptor de entrada y el modificado.
- La salida del código es un archivo "shot.txt" donde se almacena la información del descriptor de salida.
- En consola se muestra la coincidencia que tienen estos puntos.

Salida

```
Match 993: 993  
Match 994: 994  
Match 995: 995  
Match 996: 996  
Match 997: 997  
Match 998: 998
```

```
Described keypoints: 1000.  
Correct Matches: 703 out of 715.  
Matches under threshold: 285.  
Recall: 0.703000, 1-Precision: 0.016783
```