In [3]:

```
import nltk #need for dealing with text
import os #need for looping through folders
import string
import numpy as np
import copy
import pandas as pd
import pickle
import re
import math #need for computing TF-IDF score
```

In [4]:

```
title = "Used"
os.chdir("C://Users//user//mini_newsgroups//comp.graphics")
paths = []
for (dirpath, dirnames, filenames) in os.walk(str(os.getcwd())+'/'+title+'/'):
    for i in filenames:
         paths.append(str(dirpath)+str("\\")+i)
```

In [6]:

```
myfile = open(paths[1])
txt = myfile.read()
print(txt)
myfile.close()
```

```
Xref: cantaloupe.srv.cs.cmu.edu alt.3d:2141 comp.graphics:37921
Path: cantaloupe.srv.cs.cmu.edu!crabapple.srv.cs.cmu.edu!fs7.ece.cmu.edu!eur
opa.eng.gtefsd.com!gatech!swrinde!zaphod.mps.ohio-state.edu!usc!elroy.jpl.na
sa.gov!ames!olivea!uunet!mcsun!fuug!kiae!relcom!newsserv
From: alex@talus.msk.su (Alex Kolesov)
Newsgroups: alt.3d,comp.graphics
Subject: Help on RenderMan language wanted!
Message-ID: <9304051103.AA01274@talus.msk.su>
Date: 5 Apr 93 11:00:50 GMT
Sender: news-service@newcom.kiae.su
Reply-To: alex@talus.msk.su
Organization: unknown
Lines: 17

Hello everybody !

If you are using PIXAR'S RenderMan 3D scene description language for creatin
g 3D worlds, please, help me.

I'm using RenderMan library on my NeXT but there is no documentation about N
eXTSTEP version of RenderMan available. I can create very complicated scenes
and render them using surface shaders,
but I can not bring them to life by applying shadows and reflections.

As far as I understand I have to define environmental and shadows maps to pr
oduce reflections and shadows, but I do not know how to use them.

Any advises or simple RIB or C examples will be appreciated.
Thanks in advance...

---
Alex Kolesov                              Moscow, Russia.
Talus Imaging & Communications Corporation
e-mail: <alex@talus.msk.su>             (NeXT mail accepted)
.
```

In [7]:

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from collections import Counter
from num2words import num2words
```

In [8]:

```
def remove_stop_words(data):
    stop_words = stopwords.words('english')
    words = word_tokenize(str(data))
    new_text = ""
    for w in words:
        if w not in stop_words:
            new_text = new_text + " " + w
    return np.char.strip(new_text)

def remove_punctuation(data):
    symbols = "!\"#$%&()*+-./:;<=>?@[\]^_`{|}~\n"
    for i in range(len(symbols)):
        data = np.char.replace(data, symbols[i], ' ')
        data = np.char.replace(data, "  ", " ")
    data = np.char.replace(data, ',', '')
    return data


def convert_lower_case(data):
    return np.char.lower(data)


def stemming(data):
    stemmer= PorterStemmer()

    tokens = word_tokenize(str(data))
    new_text = ""
    for w in tokens:
        new_text = new_text + " " + stemmer.stem(w)
    return np.char.strip(new_text)


def convert_numbers(data):
    data = np.char.replace(data, "0", " zero ")
    data = np.char.replace(data, "1", " one ")
    data = np.char.replace(data, "2", " two ")
    data = np.char.replace(data, "3", " three ")
    data = np.char.replace(data, "4", " four ")
    data = np.char.replace(data, "5", " five ")
    data = np.char.replace(data, "6", " six ")
    data = np.char.replace(data, "7", " seven ")
    data = np.char.replace(data, "8", " eight ")
    data = np.char.replace(data, "9", " nine ")
    return data


def remove_header(data):
    try:
        ind = data.index('\n\n')
        data = data[ind:]
    except:
        print("No Header")
    return data


def remove_apostrophe(data):
    return np.char.replace(data, "'", "")

def remove_single_characters(data):
```

```
    words = word_tokenize(str(data))
    new_text = ""
    for w in words:
        if len(w) > 1:
            new_text = new_text + " " + w
    return np.char.strip(new_text)
```

In [9]:

```
def preprocess(data):
    data = remove_header(data)
    data = convert_lower_case(data)
    data = convert_numbers(data)
    data = remove_punctuation(data)
    data = remove_stop_words(data)
    data = remove_apostrophe(data)
    data = remove_single_characters(data)
    data = stemming(data)
    return data
```

In [10]:

```
processed_text = []
for i in range(len(filenames)):
    file = open(dirpath+'/'+ filenames[i], 'r', encoding='cp1250', errors='ignore')
    text = file.read().strip()
    file.close()

    processed_text.append(word_tokenize(str(preprocess(text))))
    print(processed_text)
```

```
[['recent', 'got', 'file', 'describ', 'librari', 'render', 'routin', 'call',
'sipp', 'simpl', 'polygon', 'processor', 'could', 'anyon', 'tell', 'ftp', 's
ourc', 'code', 'newest', 'version', 'around', 'also', 've', 'never', 'use',
'renderman', 'wonder', 'renderman', 'like', 'sipp', 'ie', 'librari', 'rende
r', 'routin', 'one', 'use', 'make', 'program', 'creat', 'imag', 'thank', 'jo
e', 'tham', 'joe', 'tham', 'joth', 'ersi', 'edmonton', 'ab', 'ca']]
[['recent', 'got', 'file', 'describ', 'librari', 'render', 'routin', 'call',
'sipp', 'simpl', 'polygon', 'processor', 'could', 'anyon', 'tell', 'ftp', 's
ourc', 'code', 'newest', 'version', 'around', 'also', 've', 'never', 'use',
'renderman', 'wonder', 'renderman', 'like', 'sipp', 'ie', 'librari', 'rende
r', 'routin', 'one', 'use', 'make', 'program', 'creat', 'imag', 'thank', 'jo
e', 'tham', 'joe', 'tham', 'joth', 'ersi', 'edmonton', 'ab', 'ca'], ['hell
o', 'everybodi', 'use', 'pixar', 'renderman', 'three', 'scene', 'descript',
'languag', 'creat', 'three', 'world', 'pleas', 'help', 'use', 'renderman',
'librari', 'next', 'document', 'nextstep', 'version', 'renderman', 'avail',
'creat', 'complic', 'scene', 'render', 'use', 'surfac', 'shader', 'bring',
'life', 'appli', 'shadow', 'reflect', 'far', 'understand', 'defin', 'environ
ment', 'shadow', 'map', 'produc', 'reflect', 'shadow', 'know', 'use', 'advi
s', 'simpl', 'rib', 'exampl', 'appreci', 'thank', 'advanc', 'alex', 'koleso
v', 'moscow', 'russia', 'talu', 'imag', 'commun', 'corpor', 'mail', 'alex',
'talu', 'msk', 'su', 'next', 'mail', 'accept']]
[['recent', 'got', 'file', 'describ', 'librari', 'render', 'routin', 'call',
'sipp', 'simpl', 'polygon', 'processor', 'could', 'anyon', 'tell', 'ftp', 's
ourc', 'code', 'newest', 'version', 'around', 'also', 've', 'never', 'use',
'renderman', 'wonder', 'renderman', 'like', 'sipp', 'ie', 'librari', 'rende
r', 'routin', 'one', 'use', 'make', 'program', 'creat', 'imag', 'thank', 'jo
e', 'tham', 'joe', 'tham', 'joth', 'ersi', 'edmonton', 'ab', 'ca'], ['hell
o', 'everybodi', 'use', 'pixar', 'renderman', 'three', 'scene', 'descript',
'languag', 'creat', 'three', 'world', 'pleas', 'help', 'use', 'renderman',
'librari', 'next', 'document', 'nextstep', 'version', 'renderman', 'avail',
'creat', 'complic', 'scene', 'render', 'use', 'surfac', 'shader', 'bring',
'life', 'appli', 'shadow', 'reflect', 'far', 'understand', 'defin', 'environ
ment', 'shadow', 'map', 'produc', 'reflect', 'shadow', 'know', 'use', 'advi
s', 'simpl', 'rib', 'exampl', 'appreci', 'thank', 'advanc', 'alex', 'koleso
v', 'moscow', 'russia', 'talu', 'imag', 'commun', 'corpor', 'mail', 'alex',
'talu', 'msk', 'su', 'next', 'mail', 'accept'], ['anybodi', 'know', 'good',
'two', 'graphic', 'packag', 'avail', 'ibm', 'rs', 'six', 'zero', 'zero', 'ze
ro', 'aix', 'look', 'someth', 'like', 'dec', 'gk', 'hewlett', 'packard', 'st
arbas', 'reason', 'good', 'support', 'differ', 'output', 'devic', 'like', 'p
lotter', 'termin', 'etc', 'tri', 'also', 'xgk', 'one', 'one', 'distribut',
'ibm', 'implement', 'phig', 'work', 'requir', 'output', 'devic', 'window',
'salesman', 'ibm', 'familiar', 'graphic', 'expect', 'good', 'solut', 'ari',
'ari', 'suutari', 'ari', 'carel', 'fi', 'carelcomp', 'oy', 'lappeenranta',
'finland']]
[['recent', 'got', 'file', 'describ', 'librari', 'render', 'routin', 'call',
'sipp', 'simpl', 'polygon', 'processor', 'could', 'anyon', 'tell', 'ftp', 's
ourc', 'code', 'newest', 'version', 'around', 'also', 've', 'never', 'use',
'renderman', 'wonder', 'renderman', 'like', 'sipp', 'ie', 'librari', 'rende
r', 'routin', 'one', 'use', 'make', 'program', 'creat', 'imag', 'thank', 'jo
e', 'tham', 'joe', 'tham', 'joth', 'ersi', 'edmonton', 'ab', 'ca'], ['hell
```

```
o', 'everybodi', 'use', 'pixar', 'renderman', 'three', 'scene', 'descript',
'languag', 'creat', 'three', 'world', 'pleas', 'help', 'use', 'renderman',
'librari', 'next', 'document', 'nextstep', 'version', 'renderman', 'avail',
'creat', 'complic', 'scene', 'render', 'use', 'surfac', 'shader', 'bring',
'life', 'appli', 'shadow', 'reflect', 'far', 'understand', 'defin', 'environ
ment', 'shadow', 'map', 'produc', 'reflect', 'shadow', 'know', 'use', 'advi
s', 'simpl', 'rib', 'exampl', 'appreci', 'thank', 'advanc', 'alex', 'koleso
v', 'moscow', 'russia', 'talu', 'imag', 'commun', 'corpor', 'mail', 'alex',
'talu', 'msk', 'su', 'next', 'mail', 'accept'], ['anybodi', 'know', 'good',
'two', 'graphic', 'packag', 'avail', 'ibm', 'rs', 'six', 'zero', 'zero', 'ze
ro', 'aix', 'look', 'someth', 'like', 'dec', 'gk', 'hewlett', 'packard', 'st
arbas', 'reason', 'good', 'support', 'differ', 'output', 'devic', 'like', 'p
lotter', 'termin', 'etc', 'tri', 'also', 'xgk', 'one', 'one', 'distribut',
'ibm', 'implement', 'phig', 'work', 'requir', 'output', 'devic', 'window',
'salesman', 'ibm', 'familiar', 'graphic', 'expect', 'good', 'solut', 'ari',
'ari', 'suutari', 'ari', 'carel', 'fi', 'carelcomp', 'oy', 'lappeenranta',
'finland'], ['requir', 'bgi', 'driver', 'super', 'vga', 'display', 'super',
'xvga', 'display', 'anyon', 'know', 'could', 'obtain', 'relev', 'driver', 'f
tp', 'site', 'regard', 'simon', 'crow']]
```

In [11]:

```
DF = {}
N = len(processed_text)
for i in range(N):
    tokens = processed_text[i]
    for w in tokens:
        try:
            DF[w].add(i)
        except:
            DF[w] = {i}
for i in DF:
    DF[i] = len(DF[i])
```

In [12]:

```
DF
```

Out[12]:

```
{'recent': 1,
 'got': 1,
 'file': 1,
 'describ': 1,
 'librari': 2,
 'render': 2,
 'routin': 1,
 'call': 1,
 'sipp': 1,
 'simpl': 2,
 'polygon': 1,
 'processor': 1,
 'could': 2,
 'anyon': 2,
 'tell': 1,
 'ftp': 2,
 'sourc': 1,
 'code': 1,
```

In [13]:

```
total_voca=len(DF)
print(total_voca)
```

```
144
```

In [14]:

```
def doc_freq(word):
    c = 0
    try:
        c = DF[word]
    except:
        pass
    return c
```

In [15]:

```
Word = "render"
print("the word is : ",Word,"the frequncey ",doc_freq(Word))
```

```
the word is :  render the frequncey  2
```

In [16]:

```
doc = 0
tf_idf = {}
for i in range(N):
    tokens = processed_text[i]
    counter = Counter(tokens + processed_text[i])
    words_count = len(tokens + processed_text[i])

    for token in np.unique(tokens):
        tf = counter[token]/words_count
        df = doc_freq(token)
        idf = np.log((N+1)/(df+1))
        tf_idf[doc, token] = tf*idf

    doc += 1

tf_idf
```

Out[16]:

```
{(0, 'ab'): 0.018325814637483104,
 (0, 'also'): 0.010216512475319815,
 (0, 'anyon'): 0.010216512475319815,
 (0, 'around'): 0.018325814637483104,
 (0, 'ca'): 0.018325814637483104,
 (0, 'call'): 0.018325814637483104,
 (0, 'code'): 0.018325814637483104,
 (0, 'could'): 0.010216512475319815,
 (0, 'creat'): 0.010216512475319815,
 (0, 'describ'): 0.018325814637483104,
 (0, 'edmonton'): 0.018325814637483104,
 (0, 'ersi'): 0.018325814637483104,
 (0, 'file'): 0.018325814637483104,
 (0, 'ftp'): 0.010216512475319815,
 (0, 'got'): 0.018325814637483104,
 (0, 'ie'): 0.018325814637483104,
 (0, 'imag'): 0.010216512475319815,
 (0, 'ioe'): 0.03665162927496621,
```

In [17]:

```
tf_idf[(0,'also')]
```

Out[17]:

```
0.010216512475319815
```

In [18]:

```
def matching_score(k, query):
    preprocessed_query = preprocess(query)
    tokens = word_tokenize(str(preprocessed_query))
    print("Matching Score")
    print("\nQuery:", query)
    print("")
    print(tokens)

    query_weights = {}
    for key in tf_idf:
        if key[1] in tokens:
            try:
                query_weights[key[0]] += tf_idf[key]
            except:
                query_weights[key[0]] = tf_idf[key]

    query_weights = sorted(query_weights.items(), key=lambda x: x[1], reverse=True)
    print("")

    l = []

    for i in query_weights[:k]:
        l.append(i[0])

    print(l)

matching_score(2, "I recently got a file describing a library")
```

```
No Header
Matching Score

Query: I recently got a file describing a library

['recent', 'got', 'file', 'describ', 'librari']

[0, 1]
```

In [20]:

```
title = "comp.graphics"
os.chdir(r'C://Users//user//mini_newsgroups')
paths = []
for (dirpath, dirnames, filenames) in os.walk(str(os.getcwd())+'/'+title+'/'):
    for i in filenames:
        paths.append(str(dirpath)+str("\\")+i)

processed_text = []
for i in range(len(filenames)):
    file = open(dirpath+'/'+ filenames[i], 'r', encoding='cp1250', errors='ignore')
    text = file.read().strip()
    file.close()
    processed_text.append(word_tokenize(str(preprocess(text))))


DF = {}
N = len(processed_text)

for i in range(N):
    tokens = processed_text[i]
    for w in tokens:
        try:
            DF[w].add(i)
        except:
            DF[w] = {i}

for i in DF:
    DF[i] = len(DF[i])


doc = 0
tf_idf = {}
for i in range(N):
    tokens = processed_text[i]
    counter = Counter(tokens + processed_text[i])
    words_count = len(tokens + processed_text[i])

    for token in np.unique(tokens):
        tf = counter[token]/words_count
        df = doc_freq(token)
        idf = np.log((N+1)/(df+1))
        tf_idf[doc, token] = tf*idf

    doc += 1

tf_idf
```

Out[20]:

```
{(0, 'ab'): 0.018325814637483104,
 (0, 'also'): 0.010216512475319815,
 (0, 'anyon'): 0.010216512475319815,
 (0, 'around'): 0.018325814637483104,
 (0, 'ca'): 0.018325814637483104,
 (0, 'call'): 0.018325814637483104,
 (0, 'code'): 0.018325814637483104,
 (0, 'could'): 0.010216512475319815,
 (0, 'creat'): 0.010216512475319815,
```

```
(0, 'describ'): 0.018325814637483104,
(0, 'edmonton'): 0.018325814637483104,
(0, 'ersi'): 0.018325814637483104,
(0, 'file'): 0.018325814637483104,
(0, 'ftp'): 0.010216512475319815,
(0, 'got'): 0.018325814637483104,
(0, 'ie'): 0.018325814637483104,
(0, 'imag'): 0.010216512475319815,
```

In [ ]: