Subdivision алгоритми за 3D повърхнини. Алгоритми на Catmull-Clark, Doo-Sabin и Loop

Ясен Банчев, ф.н. 80110

Общо описание на subdivision алгоритмите

Subdivision алгоритмите за повърхнини са повлияни от идеята на Чайкин за криви в равнината и първия такъв алгоритъм, този на Doo-Sabin, цели да бъде обобщение за пространството. Те дават удобен начин за получаване на гладка повърхнина по зададени контролни точки. На всяка стъпка от такъв алгоритъм се получават неколкократно повече върхове, съответно лица. По този начин се получава компактно и ефективно представяне на повърхнините като алгоритъмът може да бъде приложен произволен брой пъти за все по-добро приближение и обикновено само след няколко стъпки се получава достатъчно добър за човешкото око резултат. Удобно свойтво е и това, че при промяна на една контролна точка се променя повърхнината само в непосредствена близост до нея. Всеки от трите описани алгоритъма е апроксимиращ и C^2 гладък навсякъде освен в някои необичайни точки.

Алгоритъм на Catmull-Clark

Алгоритъмът работи за произволни модели. Има свойството, че след първата стъпка всички лица са с по 4 страни.
Описание:
За всяко лице се изчислява лицева тока – центъра на тежестта.
За всяко ребро – ребрена точка – средното на двата края на реброто и на лицевите точки на двете лица на реброто.
За всеки връх v – върхова точка – $\frac{1}{n}Q+\frac{2}{n}R+\frac{n-3}{n}v$, където
Q – средното аритметично на лицевите точки на лицата на v
R – средното аритметично на средите на ребрата с краища v
n – брой ребра с край v
При така получените точки, за всяко лице и всеки негов връх съответно се получава по едно ново лице, в което участват лицевата точка, върховата точка за този връх и двете ребрени точки на ребрата част от това лице и инцидентни с върха. Необичайни точки се получават само на първа стъпка – центровете на лица, които не са с 4 страни.
Вижда се, че точките в алгоритъма след първата си стъпка може да бъдат пресмятани с матрица на коефициентите пред необходимите точки.

Алгоритъм на Doo-Sabin

При този алгоритъм новите точки се получават за всяко лице и всеки връх в него съответно като средата на отсечката свързваща върха с центъра на тежестта на лицето. От новите точки се получават три вида лица:

- F-лица – получени от свързване на всички нови точки за някое лице
- E-лица – получени от свързване на новите точки за краищата на някое ребро за всяко от двете лица на реброто
- V-лица – получени от свързване на новите точки за всяко лице за някой връх

Ако с C_j означим връх от досегашно лице, част от модела, то новите върхове c_j се получават по формулата $c_i=\sum_{j=1}^{n} a_{ij}C_j$, където n е броя върхове в лицето и $a_{ij}=\frac{n+5}{4n}$ когато i == j и $a_{ij}=\frac{1}{4n}\left[3+2\cos\frac{2\pi(i-j)}{n}\right]$ иначе.

Тъй като E-лицата са четириъгълници и на всяка стъпка за всяко нечитириъгълно лице се получава точно едно нечитириъгълно F-лице и за всеки връх с валентност различна от 4 се получава едно V-лице, което не е с 4 страни, но всеки от върховете му е с валентност 4, то модела клони към такъв само с четириъгълници, а необичайните точки се получават точно в центровете на нечитириъгълните лица.

Алгоритъм на Loop

За разлика от предишните два този алгоритъм работи само за модели състоящи се изцяло от триъгълници.
Описание:
За всяко ребро се изчислява ребрена точка – 3/8 от сумата на двата му края + 1/8 сумата на двете точки, които участват в един и същи триъгълник с реброто.

За всеки връх v – върхна точка – $(1-n\alpha)v+\alpha\sum_{j=1}^{n} v_j$, където $v_1, v_2, \ldots, v_n$ са съседните на v върхове, а $\alpha=\frac{3}{16}$ при n == 3 и $\alpha=\frac{1}{n}\left[\frac{5}{8}-\left(\frac{3}{8}+\frac{1}{4}\cos\frac{2\pi}{n}\right)^2\right]$ при n > 3.

От всеки триъгълник се получават 4 нови. Единият – като се свържат 3-те ребрени точки, другите 3 – като се свърже върхната точка в някой връх с ребрените на двете ребра инцидентни с върха.

Реализация на приложението за визуализация

Заради по-голямото удобство е написана програма на Ruby – графичен интерфейс за управление на основното приложение, която комуникира с него посредством TCP протокола на порт 55555. Всичко останало е написано на C++. Използвани са библиотеките glut – за визуализация на OpenGL и обработване на сигнали от клавиатурата, sdl – за сокети и pthread – за многонишковост под Windows (необходима заради комуникацията чрез сокети).
За прилагането на алгоритмите първоначалната естествена форма на моделите – списък от многоъгълници се превръща в DCEL структура, с чиято помощ могат лесно да се получават съседни лица, върхове, всички ребра за дадено лице и т.н. необходими компоненти за всеки един от алгоритмите. От тази структура се генерират новополучените лица и така се получава нов модел, като най-често се итерират всички лица в структурата, всички ребра и всички върхове за да се изчислят новите лица, ребра и върхове.
Описание на основните класове и функции в тях:

- Файловете **mesh_parser.h** и **mesh_parser.cpp** съдържат реализацията на зареждането на двата поддържани формата за модели - **OFF** и **STL**. Има общ интерфейс за всеки от форматите - **class MeshParser**.
- **class Point** – точка в пространството, над нея са дефинирани и естествените аритметични операции
- **class MeshFace –** основно съдържа списък от **Point –** лице част от модел и функция **CalculateNormal –** за изчисляване на нормалата към лицето (необходимо при изчисляване на осветяването на обекта).
- Във файловете **dcel.h** и **dcel.cpp** може да бъде видяна реализацията на DCEL структурата. Всеки тип (връх, ребро, лице) съдържа и индекса под който се намира в списъка с върхове, ребра, лица.
- **class SubdivisionAlgorithm** – интерфейсен клас, който наследяват всички класове за конкретни алгоритми. Всички споделят общия метод **BuildDCEL**, който построява DCEL структура по даден модел и всеки от касловете **CatmullClark**, **DooSabin** и **Loop** има своя реализация на метода **NextStep –** резултата от прилагане на алгоритъма еднократно.
- **class Subdivisor** – използвайки инстанция на клас от тип **SubdivisionAlgorithm** получава модел след прилагането на даден брой стъпки от алгоритъма, като ако броя стъпки е с 1 по-голям от преишната, за която е получен модел се прилага само 1 стъпка от алгоритъма, ако не – се изчислява наново от стъпка 0 до дадената.

- **class Visualizer –** съдържа всички инициализационни стъпки, комуникацията с Ruby приложението, и визуализацията с помощта на OpenGL. Управлява **class Subdivisor** и изобразява получения от него модел. При зареждането на модел чрез класа **Parser** се взима обхвата на координатите на върховете с цел мащабиране и горе-долу еднаква големина на различните модели на екрана.

Всеки от алгоритмите е реализиран следвайки описанието в предишната част на тази документация. В случаите, в които някоя нова точка не е дефинирана поради липсата на някой елемент (най-често когато модела не е затворен) на нейно място като маркер се поставя несъществуваща точка с координати максималните програмно допустими координати.
За съжаление не е реализирана предварителна триангулация за моделите, които го изискват преди прилагането на алгоритъма на Loop и съответно той работи само за малка част от приложените модели.

Указания за ползване

Тъй като приложението се състои от две отделни програми – едната основна и написана на C++ правеща всички изчисления и визуализация и една, написана на Ruby, която служи за управление на предишната, е необходимо приложението да се стартира чрез файла **start.bat**.
От прозореца за управление се избира първо един модел, който е във формат **STL** или **OFF** и се намира в папката **meshes**. Избира се някой от алгоритмите и се задава коя стъпка от алгоритъма да се визуализира. При избиране на нов модел стпъката се рестартира на 0, но при избиране на нов алгоритъм се запазва, заедно с ориентацията на модела с цел паралелна демонстрация на алгоритмите. Допълнително може да се зададе дали модела да бъде показан като плътен обект или мрежа (wireframe) и ако да с каква дебелина на линията, дали да бъде осветен и цвят.
Позиционирането и големината на модела се управляват от прозореца за визуализация с помощта на клавиатурата. Лява и дясна стрелка завъртат обекта спрямо неговата абсциса **Oy**, горна и долна – **Ox**, а **z** и **x – Oz**. C + и **–** обектът се приближава и отдалечава.