Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Робототехника и комплексная автоматизация

КАФЕДРА Системы автоматизированного проектирования (РК-6)

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

по дисциплине: «Компьютерная графика»

Студент	Гусаров Аркадий Андреевич
Группа	РК6-63Б
Тип задания	Лабораторная работа №6-7
Название	«Моделирование водяного фильтра»

Студент _____ **Гусаров А.А.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Витюков Ф.А.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2022 г.

ОГЛАВЛЕНИЕ

ЦЕЛЬ РАБОТЫ

Цель лабораторной работы – продолжить изучение базовых функций PhysX. Реализовать визуализацию гранульного фильтра и пропуска через него молекул воды. Реализовать подключение настроечного файла в формате xml для настройки основных параметров программы.

Задание

На основе PhysX Tutorials реализовать с использованием уже имеющихся элементов визуализацию создания гранульного фильтра и пропуска через него молекул воды.

Для этого задачу декомпозировать на следующие:

а) Создание фильтра: гранулы фильтра – шары с диаметром sphereDiameter со случайно заданным отклонением в пределах sphereDiameterTolerance. Необходимо сгенерировать «облако» таких шаров (в целом может представлять собой кубическую форму), расположенное над формой для фильтра. Шары, падая под действием гравитации, будут засыпаться в фильтр.

б) Генерация формы для фильтра: внутри – цилиндр с диаметром innerCylinderD. Внешняя оболочка диаметра outerCylinderD представляет собой полый открытый цилиндр без верхнего круга. Высота цилиндров cylinderH. cylindersDelta – расстояние между основаниями цилиндров (внутрений цилиндр приподнят относительно внешнего).

Функция простановки внутреннего цилиндра в PhysX Tutorials есть, а для генерации внешней оболочки предстоит написать свой алгоритм. При этом нужно помнить, что поверхность цилиндра в обычном виде обращена к пользователю внешней частью. Внешняя часть – рабочая, она рендерится на экране. Внутренняя же часть «прозрачна» (невидима). Направление обхода точек треугольника (против или по часовой стрелке) определяет нормаль к нему, а, следовательно, видимость треугольника.

В вашем случае при генерации полого цилиндра необходимо «внешнюю» часть повернуть внутрь, к оси цилиндра, изменив функцию генерации цилиндра (за счёт изменения в обходе соединяемых точек). в) При старте программы гранулы засыпаются в форму.

Фиксируем засыпанные гранулы в их текущем положении. Удаляем форму.

В настроечном файле, представленном в формате XML, должны быть доступны следующие параметры:

sphereDiameter

sphereDiameterTolerance

innerCylinderD

outerCylinderD

cylinderH

cylindersDelta

scaleFactor

particlesH

Вводная часть

Для создания молекул воды, то есть сфер, необходимо установить размеры облака, из которого будут появляться эти молекулы. Размеры облака задаются параметрами из настроечного файла. Далее определяются координаты всех сфер в заданном облаке и в этих точках создаются сферы.

Поверхность фильтра состоит из двух цилиндров, помещенных друг в друга. Поверхности цилиндров будут замещаться треугольниками, составленными из вершин.

Данные вершины будут располагаться по окружностям разных радиусов (внутренний и внешний цилиндр).

С помощью клавиши клавиатуры «space bar» (пробел) будет осуществляться фиксирование засыпанных гранул (тех, которые находятся в объеме фильтра) в их текущем положении. Также будет реализовано удаление

оставшихся вне фильтра молекул. Данный функционал будет реализован с помощью обработки сигналов.

Считывание файла с настройками реализовано с помощью библиотеки tinyxml. В программе осуществляется обращение к файлу config.xml и производится считывание по тегам. Полученная из тегов информация записывается в переменные, отвечающие за переменные конфигурации программы.

Разбор кода

С помощью функции CreateSphere, принимающей набор координат центра сферы, реализуется создание молекулы. Данная функция возвращает переменную типа NxActor*, то есть ссылку на созданный объект.

```
393  NxActor* CreateSphere(NxVec3 s)
394  {
395      // Set the sphere starting height to 3.5m so box starts off falling onto the ground
396      NxReal sphereStartHeight = 0;
397      NxReal start_box_posX = s[0];
398      NxReal start_box_posY = s[1];
399      NxReal start_box_posZ = s[2];
400
401      // Add a single-shape actor to the scene
402      NxActorDesc actorDesc;
403      NxBodyDesc bodyDesc;
404
405      // The actor has one shape, a sphere, 1m on radius
406      NxSphereShapeDesc sphereDesc;
407      sphereDesc.radius = sphereDiameter / 2 + (float(rand() % int(sphereDiameterTolerance * 100))) / 100 - (float(rand() % int(sphereDiameterTolerance * 100))) / 100;
408      sphereDesc.localPose.t = NxVec3(0, 0, 0);
409
410      actorDesc.shapes.pushBack(&sphereDesc);
411      actorDesc.body = &bodyDesc;
412      actorDesc.density = 1.0f;
413      actorDesc.globalPose.t = NxVec3(start_box_posX, start_box_posY, start_box_posZ);
414      return gScene->createActor(actorDesc);
415  }
```

С помощью функции CreateCloudSphere реализуется создание облака гранул. В данной функции подсчитывается количество сфер, исходя из размеров облака и диаметра сфер.

```
417  void CreateCloudSphere()
418  {
419      int count_x_sphere = outerCylinderD / (sphereDiameter);
420      int count_y_sphere = particlesH;
421      int count_z_sphere = count_x_sphere;
422      int startX = -count_x_sphere * sphereDiameter / 2;
423      int startY = 3 * cylinderH;
424      int startZ = startX;
425      int count = 0;
426
427      count_of_sphere = (count_y_sphere)*(count_z_sphere)*(count_x_sphere);
428      spheres_pos = new float*[count_of_sphere];
429      sphere = new NxActor*[count_of_sphere];
430
431      for (int i = 0; i < count_of_sphere; i++)
432          spheres_pos[i] = new float[3];
433      srand(time(NULL));
434
435      for (int i = 0; i < count_x_sphere; i++)
436          for (int j = 0; j < count_y_sphere; j++)
437              for (int k = 0; k < count_z_sphere; k++)
438              {
439                  spheres_pos[count][0] = startX + k * (sphereDiameter + sphereDiameterTolerance);
440                  spheres_pos[count][1] = startY + j * (sphereDiameter + sphereDiameterTolerance);
441                  spheres_pos[count][2] = startZ + i * (sphereDiameter + sphereDiameterTolerance);
442                  count++;
443              }
444
445      for (int i = 0; i < count_of_sphere; i++)
446          sphere[i] = CreateSphere(NxVec3(spheres_pos[i][0], spheres_pos[i][1], spheres_pos[i][2]));
447  }
```

С помощью функции CreateInnerCylinder выполняется создание внутреннего цилиндра. Сначала производится генерация вершин, записываются вершины нижнего и верхнего оснований с заданным интервалом. Далее определяется последовательность индексов вершин таким образом, чтобы нормали треугольников были направлены к центру. После этого выполняется создание поверхности и создание треугольной сетки.

```
471  NxActor* CreateInnerCylinder()
472  {
473      NxActorDesc actorDesc;
474      NxBodyDesc bodyDesc;
475      bodyDesc.flags |= NX_BF_KINEMATIC;
476
477      // Pyramid
478      NxVec3 verts[48];
479
480      for (int i = 0, j = 0; i < 24; i++, j = j + 15)
481      {
482          verts[i] = NxVec3(innerCylinderD / 2 * cos(j * PI / 180), cylindersDelta, innerCylinderD / 2 * sin(j * PI / 180));
483          verts[i + 24] = NxVec3(innerCylinderD / 2 * cos(j * PI / 180), cylinderH, innerCylinderD / 2 * sin(j * PI / 180));
484      }
485
486
487      // Create descriptor for convex mesh
488      if (!convexDesc)
489      {
490          convexDesc = new NxConvexMeshDesc();
491          assert(convexDesc);
492      }
493      convexDesc->numVertices = 48;
494      convexDesc->pointStrideBytes = sizeof(NxVec3);
495      convexDesc->points = verts;
496      convexDesc->flags = NX_CF_COMPUTE_CONVEX;
497
498      NxConvexShapeDesc convexShapeDesc;
499      convexShapeDesc.localPose.t = NxVec3(0, 0, 0);
500      convexShapeDesc.userData = convexDesc;
501
502      // Two ways on cooking mesh: 1. Saved in memory, 2. Saved in file
503      NxInitCooking();

505      // Cooking from memory
506      MemoryWriteBuffer buf;
507      bool status = NxCookConvexMesh(*convexDesc, buf);
508
509      NxConvexMesh *pMesh = gPhysicsSDK->createConvexMesh(MemoryReadBuffer(buf.data));
510      assert(pMesh);
511      convexShapeDesc.meshData = pMesh;
512      NxCloseCooking();
513
514      if (pMesh)
515      {
516          // Save mesh in userData for drawing.
517          pMesh->saveToDesc(*convexDesc);
518          NxActorDesc actorDesc;
519          assert(convexShapeDesc.isValid());
520          actorDesc.shapes.pushBack(&convexShapeDesc);
521          actorDesc.body = &bodyDesc;
522          actorDesc.density = 1.0f;
523
524          actorDesc.globalPose.t = NxVec3(0.0f, 0.0f, 0.0f);
525          assert(actorDesc.isValid());
526          NxActor* actor = gScene->createActor(actorDesc);
527          assert(actor);
528          return actor;
529      }
530
531      return NULL;
532  }
```

С помощью функции CreateOuterCylinder выполняется построение внешнего цилиндра.

```
534 NxActor* CreateOuterCylinder()
535 {
536     // Supply hull
537     NxVec3 verts[48];
538     for (int i = 0, j = 0; i < 24; i++, j = j + 15)
539     {
540         verts[i] = NxVec3(outerCylinderD / 2 * cos(j * PI / 180), 0, outerCylinderD / 2 * sin(j * PI / 180));
541         verts[i + 24] = NxVec3(outerCylinderD / 2 * cos(j * PI / 180), cylinderH, outerCylinderD / 2 * sin(j * PI / 180));
542     }
543
544     // Triangles is 12*3
545     NxU32 indices[48 * 3] =
546     { 1, 24, 0,
547         1, 25, 24,
548         2, 25, 1,
549         2, 26, 25,
550         3, 26, 2,
551         3, 27, 26,
552         4, 27, 3,
553         4, 28, 27,
554         5, 28, 4,
555         5, 29, 28,
556         6, 29, 5,
557         6, 30, 29,
558         7, 30, 6,
559         7, 31, 30,
560         8, 31, 7,
561         8, 32, 31,
562         9, 32, 8,
563         9, 33, 32,
564         10, 33, 9,
565         10, 34, 33,
566         11, 34, 10,
567         11, 35, 34,
568         12, 35, 11,
569         12, 36, 35,
570         13, 36, 12,
571         13, 37, 36,
572         14, 37, 13,
573         14, 38, 37,

574         15, 38, 14,
575         15, 39, 38,
576         16, 39, 15,
577         16, 40, 39,
578         17, 40, 16,
579         17, 41, 40,
580         18, 41, 17,
581         18, 42, 41,
582         19, 42, 18,
583         19, 43, 42,
584         20, 43, 19,
585         20, 44, 43,
586         21, 44, 20,
587         21, 45, 44,
588         22, 45, 21,
589         22, 46, 45,
590         23, 46, 22,
591         23, 47, 46,
592         0, 47, 23,
593         0, 24, 47,
594     };
595     if (!triangleMeshDesc)
596     {
597         triangleMeshDesc = new NxTriangleMeshDesc();
598         assert(triangleMeshDesc);
599     }
600     triangleMeshDesc->numVertices = 48;
601     triangleMeshDesc->pointStrideBytes = sizeof(NxVec3);
602     triangleMeshDesc->points = verts;
603     triangleMeshDesc->numTriangles = 48;
604     triangleMeshDesc->flags = 0;
605     triangleMeshDesc->triangles = indices;
606     triangleMeshDesc->triangleStrideBytes = 3 * sizeof(NxU32);
607
608
609     // The actor has one shape, a triangle mesh
610     NxInitCooking();
611     MemoryWriteBuffer buf;
```

```
613     bool status = NxCookTriangleMesh(*triangleMeshDesc, buf);
614     NxTriangleMesh* pMesh;
615     if (status)
616     {
617         pMesh = gPhysicsSDK->createTriangleMesh(MemoryReadBuffer(buf.data));
618     }
619     else
620     {
621         assert(false);
622         pMesh = NULL;
623     }
624     NxCloseCooking();
625     // Create TriangleMesh above code segment.
626
627     NxTriangleMeshShapeDesc tmsd;
628     tmsd.meshData = pMesh;
629     tmsd.userData = triangleMeshDesc;
630     tmsd.localPose.t = NxVec3(0, 0, 0);
631     tmsd.meshPagingMode = NX_MESH_PAGING_AUTO;
632
633     NxActorDesc actorDesc;
634     NxBodyDesc  bodyDesc;
635     assert(tmsd.isValid());
636     actorDesc.shapes.pushBack(&tmsd);
637     //Dynamic triangle mesh don't be supported anymore. So body = NULL
638     actorDesc.body = NULL;
639     //actorDesc.density = 10.0f;
640     actorDesc.globalPose.t = NxVec3(0.0f, 0.0f, 0.0f);
641
642     if (pMesh)
643     {
644         // Save mesh in userData for drawing
645         pMesh->saveToDesc(*triangleMeshDesc);
646         //
647         assert(actorDesc.isValid());
648         NxActor* pActor = gScene->createActor(actorDesc);
649         assert(pActor);
650
651         return pActor;
652     }
653
654     return NULL;
655 }
```

С помощью функции get_config реализуется считывание с файла config.xml таких параметров, как: sphereDiameter, sphereDiameterTolerance, innerCylinderD, outerCylinderD, cylinderH, cylindersDelta, particlesH.

```
783  void get_config()
784  {
785      TiXmlDocument doc("config.xml");
786      TiXmlElement* xml_root = 0;
787      TiXmlElement* xml_curr = 0;
788
789      if (!doc.LoadFile())
790      {
791          printf("%s\n", doc.ErrorDesc());
792          return;
793      }
794
795      TiXmlHandle docHandle(&doc);
796      xml_root = docHandle.FirstChildElement("root").ToElement();
797
798      xml_curr = xml_root->FirstChildElement("sphereDiameter");
799      sphereDiameter = atof(xml_curr->GetText());
800
801      xml_curr = xml_root->FirstChildElement("sphereDiameterTolerance");
802      sphereDiameterTolerance = atof(xml_curr->GetText());
803
804      xml_curr = xml_root->FirstChildElement("innerCylinderD");
805      innerCylinderD = atof(xml_curr->GetText());
806
807      xml_curr = xml_root->FirstChildElement("outerCylinderD");
808      outerCylinderD = atof(xml_curr->GetText());
809
810      xml_curr = xml_root->FirstChildElement("cylinderH");
811      cylinderH = atof(xml_curr->GetText());
812
813      xml_curr = xml_root->FirstChildElement("cylindersDelta");
814      cylindersDelta = atof(xml_curr->GetText());
815
816      xml_curr = xml_root->FirstChildElement("particlesH");
817      particlesH = atof(xml_curr->GetText());
818  }
```

Результаты работы программы

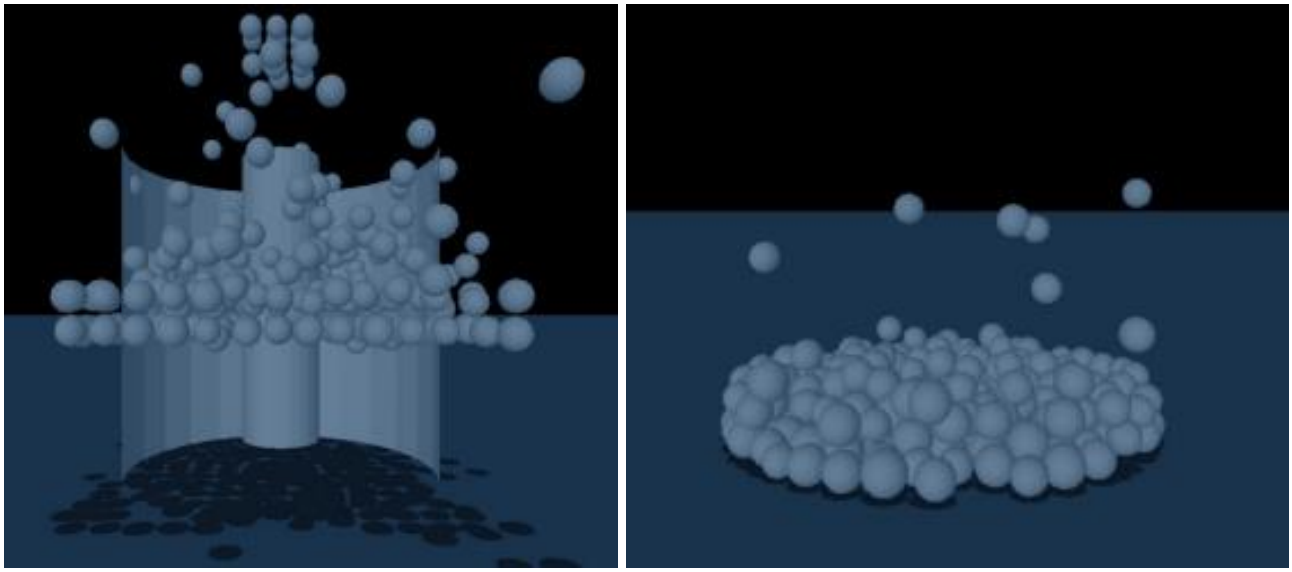

Рисунок 1. Полученная визуализация

ВЫВОДЫ

При выполнении лабораторной работы было продолжено изучение базовых функций PhysX Tutorial. Были усвоены основные принципы работы с

объектами. Также были изучены способы моделирования цилиндров и сфер, реализовано подключение настроечного файла в формате xml.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. NVIDIA PhysX SDK 4.1 Documentation [Электронный ресурс] – Режим доступа:https://gameworksdocs.nvidia.com/PhysX/4.1/documentation/physxguide/Index.html.