

КЛИЕНТ-СЕРВЕРНАЯ СИСТЕМА ВИЗУАЛИЗАЦИИ БОЛЬШИХ ОБЪЕМОВ ТРЕХМЕРНЫХ ДАННЫХ СУПЕРКОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ

Е. О. Кривошеин¹, Н. В. Снытников²

¹Новосибирский государственный университет, 630090, Новосибирск

²Институт вычислительной математики и математической геофизики СО РАН, 630090, Новосибирск

УДК 517.968

Разработан программный комплекс для визуализации данных трехмерного моделирования, представляющих собой облака точек большого размера. Он имеет клиент-серверную программную архитектуру и может быть использован для различных приложений: от архитектурно-строительных САПР до обработки данных суперкомпьютерного моделирования в задачах астрофизики или физики плазмы. Работоспособность комплекса демонстрируется на решении задачи контроля соответствия строительного объекта его проектной документации.

Ключевые слова: Облака точек, октодеревья, распределенные системы, системы автоматизированного проектирования

Введение

При суперкомпьютерном моделировании астрофизических процессов [1] или решении задач инженерного проектирования [2] (например, лазерного сканирования строительных объектов) возникает необходимость работы с большими объемами трехмерных данных — облаками точек (набором точек, каждая из которых определяется пространственной координатой и дополняется значением цвета или плотности). Общим для этих задач являются необходимость визуализации миллиардов точек, обеспечение возможности распределенного хранения, одновременного доступа и синхронизированной обработки несколькими пользователями в режиме общего редактирования. При этом требования к аппаратному обеспечению клиентских систем должны быть невысокими, чтобы иметь возможность доступа к визуализации данных с мобильного устройства (имеющего ARM процессор и 1 ГБ оперативной памяти) в сети с пропускной способностью 100 Мбит.

Подавляющая часть коммерческих и свободно распространяемых программных продуктов для работы с облаками точек являются однопользовательскими системами, требующими установку и работу с приложением на компьютере (ПК или мобильном устройстве), например, [4, 5, 6], для которых невозможно организовать одновременный доступ к модели несколькими пользователями. Однако в последние 2-3 года, в связи с бурным развитием клиент-серверных технологий, стали появляться клиент-серверные программные решения, предоставляющие доступ к модели, расположенной на сервере, непосредственно из браузера пользователя (с использованием технологии WebGL, поддерживаемой во всех современных браузерах: на ПК, смартфонах и планшетах). Среди них наиболее заметным некоммерческим проектом являются *potree.org* [7] и коммерческий сервис *PointCloudViz* [8]. В качестве основной структуры данных в них используется модифицируемое вложенное октодерево с уровнями детализации для эффективной пересылки данных и отрисовки, а также адаптирующимся под уровень размер точек для обеспечения гладкости изображения. Вместе с тем, программная архитектура подобных клиент-серверных решений предназначена для работы с предварительно подготовленными моделями (облаками точек) для визуализации данных на стороне пользователя, не позволяя модифицировать их в клиенте (браузере) во время работы, что ограничивает применимость для задач САПР.

В данной статье мы описываем программную архитектуру, основные алгоритмы и реализацию распределенной клиент-серверной системы: облака точек, структурированные в виде октодеревя, хранятся непосредственно на сервере, по запросу пользователя происходит модификация данных (если требуется пользователю), передача необходимых данных на клиентскую машину и их дальнейшая визуализация в веб-браузере с выбором необходимой для пользователя степени детализации.

Работоспособность созданной программной системы демонстрируется на решении задачи контроля соответствия строительного объекта его проектной документации (Рис. 1). Такой контроль используется в реальной практике для выявления ошибок на ранних стадиях и отслеживании процесса строительства [3]. Его суть заключается в том, что для строящегося объекта (жилого дома или производственного помещения) выполняется регулярное трехмерное лазерное сканирование с помощью тахеометров или лидаров. В качестве выходных данных сканеры выдают облако точек (с числом точек в одном скане порядка сотни миллионов). Далее эти облака точек необходимо совместить с исходной проектной цифровой моделью строительного объекта и определить дефекты — разницу (расстояние) между данными сканирования и исходной моделью. Процедура совмещения включает в себя сдвиг и поворот, необходимые для минимизации разности расстояний между точками облака и проектной 3D модели. В существующих САПР точное совмещение выполняется в два этапа: предварительный (приближенный) и точный, основанный на алгоритме ICP (Iterative Closest Point, итеративная ближайшая точка) [9] для нахождения локального оптимального смещения.

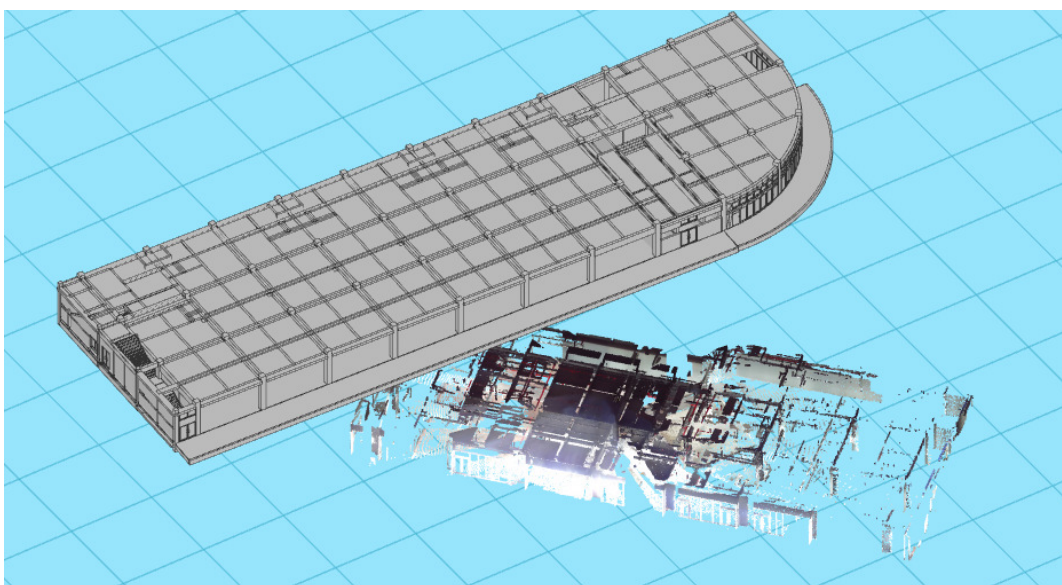


Рис. 1: 3D модель и соответствующее облако точек

1 Программная архитектура клиент-серверного приложения

Программная архитектура клиент-серверного приложения схематично представлена на Рис. 2.

Серверная часть представляет собой три основных модуля: веб-сервер, включающий в себя веб-интерфейс, обрабатывающий запросы пользователя, сервер-обработчик геометрических данных (облаков точек и цифровой модели), написанный на языке C++, и хранилище данных, включающее в себя СУБД с пользовательскими цифровыми моделями и загруженными облаками точек. При начале работы пользователь запускает веб-браузер (который является клиентским приложением) и подключается к веб-серверу по HTTP протоколу. После выбора конкретной цифровой модели из базы данных выполняется ее запуск на сервере-обработчике и устанавливается прямое соединение между сервером-обработчиком и клиентским приложением с использованием протокола WebSocket, обеспечивающего возможность непрерывного и двустороннего сообщения между клиентом и сервером.

Сервер-обработчик должен предоставлять возможность реализации широкого спектра алгоритмов по работе с облаками точек. Так, для решения задачи сравнения актуального состояния строительного объекта с проектной цифровой моделью (Рис. 3) выполняются следующие шаги:

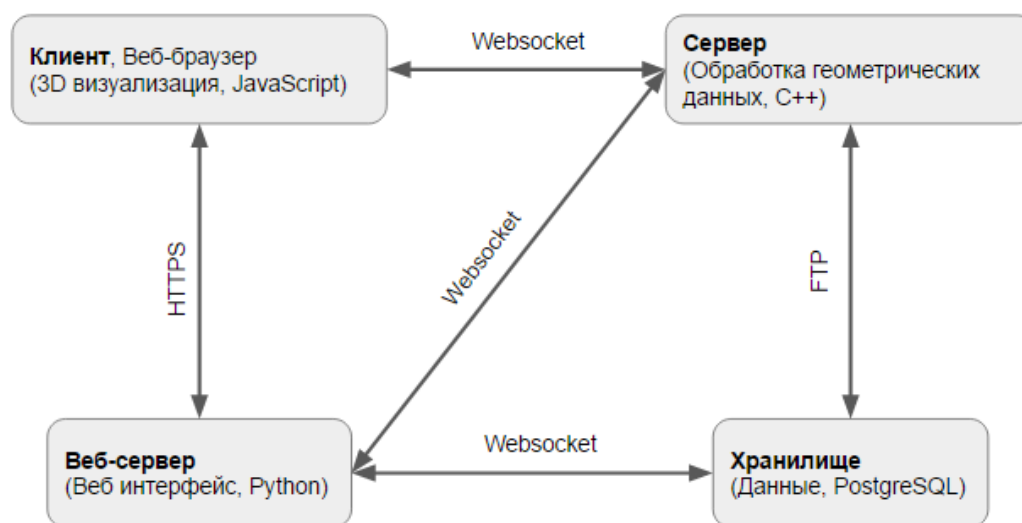


Рис. 2: Программная архитектура клиент-серверного приложения

- загрузка облака точек (результатов лазерного сканирования) на сервер в виде популярных форматов *.xyz* и *.ply*,
- декомпозиция облаков точек, позволяющая держать в памяти сервера только актуальное для пользователя подмножество точек (на основе структур данных *octree*, восьмеричного дерева), в результате чего точки в облаке распределяются между ограничивающими объемами (вокселями), и поиск ближайших соседей выполняется за логарифмическую сложность (а не квадратичную в случае полного пересчета всех расстояний между точками),
- клиент-серверная передача данных для визуализации, обеспечивающая визуализацию только актуальных для пользователя точек,
- фильтрация облака точек для удаления шумов,
- расчет нормалей и коэффициентов кривизны для каждой из точек, а также вычисление локальных признаков (предполагаемых поверхностей),
- приближенное выравнивание облака точек относительно исходной цифровой модели,
- точное выравнивание,
- расчет расстояний от точек облака до 3D модели.

2 Структура хранения облаков точек

Реализованная структура данных основана на октодереве, которое представляет собой рекурсивную сетку, где на каждом уровне 8 ячеек организованы в прямоугольный параллелепипед, — «ограничивающий объем» (*bounding box*, *BB*) облака точек. Размер *BB* равен двум ячейкам вдоль каждой оси: на каждом уровне пространство разбивается на две равные части. Во вложенном октодереве все узлы содержат точки — и внутренние, и листовые. Распределение точек по уровням октодереве позволяет подгружать их частями по необходимости, это в свою очередь дает возможность реализовать уровни детализации без дублирования точек. Точки хранятся в файлах на диске, для каждого узла вложенного октодереве — свой файл. Уровни детализации организованы через последовательную подгрузку узлов, начиная с корня октодереве. При отрисовке все узлы, попадающие в область видимости, подгружаются с диска в память. Пользователь может проходить через облако и «на лету» получать актуальные точки. Благодаря организации хранения облака точек через иерархию файлов, клиент может запрашивать узлы, как по *HTTP* через *URL*, так и по бинарным каналам (*WebSocket*).

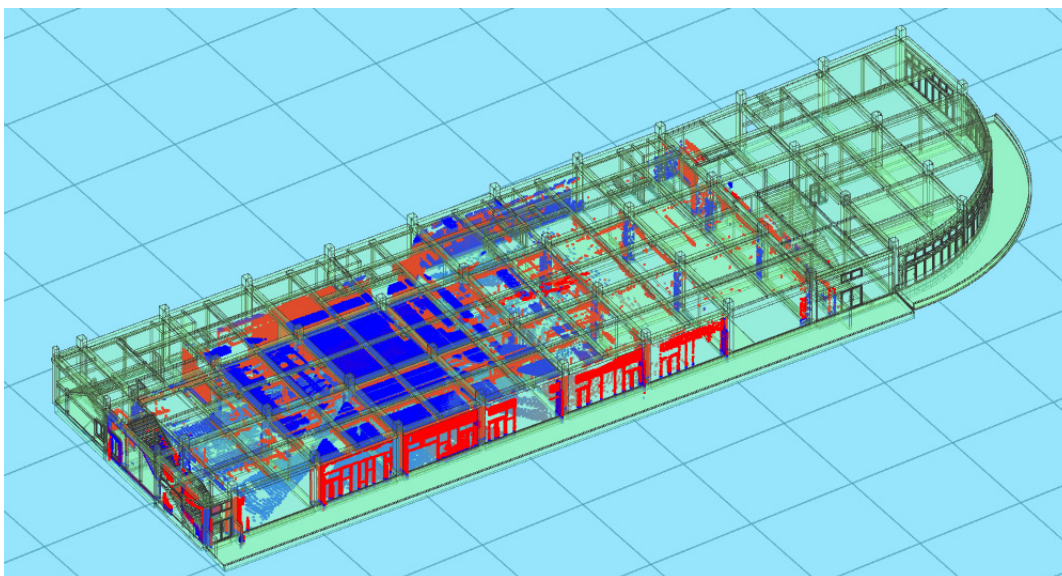


Рис. 3: Совмещенные 3D модель и облако точек. Точки окрашены в соответствии с их расстояниями до модели

Для построения октодеревя на вход подается список трехмерных точек и ВВ всего облака точек; на выходе алгоритма построения получается вложенное октодерево. Иерархия строится таким образом, чтобы количество точек в узлах было приблизительно одинаковым. Построение выполняется в два этапа: заполнение листов октодеревя и распределение точек между всеми уровнями в октодерева. На этапе заполнения (Рис. 2) точки складываются в листья октодеревя: первые *nodeCapacity* точек попадают в корень октодеревя, если в корне оказывается больше *nodeCapacity* точек — все точки корня распределяются между 8 детьми, теперь в октодерева 8 листьев и последующие точки добавляются в листья. В каждый момент времени на этапе заполнения точки находятся только в листьях октодеревя. Таким образом, в конце этапа заполнения построено октодерево, в котором точки хранятся только в листовых узлах, а промежуточные узлы остаются пустыми. При таком построении нет необходимости держать точки в памяти, поэтому при количестве точек в дереве, большего заданного параметра *maxMemory*, точки перемещаются на диск, в памяти остается только пространственная иерархия октодеревя. Благодаря этому алгоритм способен обрабатывать облака точек, не упираясь в ограничение по памяти и обеспечивает требование на работу с облаками размером 10^9 точек.

На втором этапе точки из узлов октодеревя распределяются равномерно между всеми уровнями октодеревя (Рис. 2).

Функция $Rand(min, max)$ моделирует случайную величину с плотностью вероятности:

$$f(x) = \frac{8^x}{\sum_{n=0}^{depth} 8^n} \quad (1)$$

Где *depth* — глубина октодеревя. Такая функция выбрана для равномерного распределения точек между узлами вложенного октодеревя.

3 Визуализация

При визуализации октодерева обходится в ширину, начиная с корня, для нахождения узлов, находящихся в области видимости камеры и имеющих достаточный видимый размер (Рис. 3).

Для этого ограничивающий объем узла проецируется на плоскость экрана, и если характерный размер проекции меньше некоторого заданного значения (например, 1 пиксель), узел игнорируется при отрисовке текущего кадра. На входе алгоритма задается бюджет в виде количества точек, которые необходимо визуализировать. Этот параметр позволяет пользователю контролировать уровень детализации облака и,

Алгоритм 1 Заполнение октодерева

```

1: function FILLOCTREE(pointCloud, rootBoundingBox, nodeCapacity, maxMemory)
2:   octree  $\leftarrow$  octree  $\cup$  rootBoundingBox
3:   for p in pointCloud do
4:     node  $\leftarrow$  GetLeafNodeContaining(octree, p)
5:     if size(node) + 1 < nodeCapacity then
6:       node  $\leftarrow$  node  $\cup$  p
7:     else
8:       children  $\leftarrow$  GetChildren(node)
9:       octree  $\leftarrow$  octree  $\cup$  children
10:      for point in node do
11:        idx  $\leftarrow$  GetSpatialIndex(node, point)
12:        children[idx]  $\leftarrow$  children[idx]  $\cup$  point
13:      node  $\leftarrow$   $\emptyset$ 
14:      if size(octree) > maxMemory then
15:        SavePointsToDisk(octree)
16:  return octree

```

Алгоритм 2 Распределение точек по узлам октодерева

```

1: function DISTRIBUTEPOINTS(octree)
2:   for node in octree do
3:     LoadPointsFromDisk(node)
4:     parents  $\leftarrow$  GetParents(node)
5:     for point in node do
6:       r  $\leftarrow$  Rand(0, size(parents))
7:       parents[r]  $\leftarrow$  parents[r]  $\cup$  point
8:   return octree

```

следовательно, производительность отрисовки. Если количество точек в узле, добавленное к общему количеству видимых точек, не превышает бюджета, то узел добавляется в бинарную кучу (binary heap). Бинарная куча позволяет эффективно извлекать элементы по приоритету. Приоритет узла обратно пропорционален весу $weight = Distance(node, camera)$, вычисляемому как расстояние от узла до камеры, спроецированное на прямую, перпендикулярную плоскости экрана и проходящую через его центр. Если расстояние меньше характерного размера узла, то назначается максимальный приоритет. Узлы, которые попали в бинарную кучу для визуализации, но не присутствуют в памяти клиента, должны быть запрошены с сервера. Сервер в ответ на запрос возвращает бинарный массив с точками, который будет без дополнительной обработки загружен в память видео-процессора клиента. Все подгруженные узлы попадают в кэш («Least Recently Used») и будут переиспользованы без подгрузки при очередном попадании в область видимости.

Заключение

Была разработана клиент-серверная система для визуализации облаков точек больших объемов. Представлена общая схема программной архитектуры и алгоритмы для построения структур данных для хранения облаков точек на сервере, а также для их визуализации на клиенте. Реализованные алгоритмы с механизмом уровней детализации обеспечивают возможность интерактивной навигации по облаку точек в реальном времени. Работоспособность системы протестирована на реальных промышленных данных лазерного ска-

Алгоритм 3 Алгоритм визуализации

```

1: function RENDER(root, camera, budget)
2:   numVisiblePoints  $\leftarrow$  0
3:   visibleNodes  $\leftarrow$   $\emptyset$ 
4:   loadingQue  $\leftarrow$   $\emptyset$ 
5:   binHeap  $\leftarrow$  binHeap  $\cup$  root
6:   while size(binHeap) > 0 do
7:     node  $\leftarrow$  Pop(binHeap)
8:     if IsInsideFrustum(camera, node) = false then
9:       continue
10:    if numVisiblePoints + size(node) > budget then
11:      continue
12:    numVisiblePoints  $\leftarrow$  numVisiblePoints + size(node)
13:    if IsLoaded(node) then
14:      visibleNodes  $\leftarrow$  visibleNodes  $\cup$  node
15:    else
16:      loadingQue  $\leftarrow$  loadingQue  $\cup$  node
17:    for child in node do
18:      if ProjectedOnScreen(node) < 1 then
19:        continue
20:      weight  $\leftarrow$  Distance(node, camera)
21:      if weight < BoundingSphereRadius(node) then
22:        weight  $\leftarrow$   $\infty$ 
23:      binHeap[ $\frac{1}{weight}$ ]  $\leftarrow$  node
24:  return visibleNodes

```

нирования строительных объектов (с десятками миллионов точек в одном скане) на примере решения задачи приближенного выравнивания, которое обеспечивает автоматическое выравнивание облака точек и строительного объекта.

Список литературы

- [1] Снытников Н.В. Масштабируемый параллельный алгоритм для моделирования трехмерной динамики гравитирующих систем методом частиц // Вестник УГАТУ. 2016. Т. 20. С. 137–142.
- [2] А.Г. Ершов, И.А. Рыков, Н.В. Снытников. Как создается инженерное наукоемкое ПО мирового класса // Наука из первых рук, номер 2 (50). 2013. С. 80–95.
- [3] Burcu Akinci, Frank Boukamp, Chris Gordon, Daniel Huber, Catherine Lyons, and Kuhn Park. A formalism for utilization of sensor systems and integrated project models for active construction quality control // Automation in Construction, 2006, 15(2), pp. 124–138
- [4] Point Cloud Library // <http://www.pointclouds.org>
- [5] Geomagic 3D Scanning and Reverse Engineering Software // <http://www.geomagic.com/en/products-landing-pages/scanning>
- [6] Trimble Realworks // <https://geospatial.trimble.com/products-and-solutions/trimble-realworks>
- [7] Potree: Rendering Large Point Clouds in Web Browsers // <http://www.potree.org>

- [8] PointCloudViz (Mirage Technologies: point cloud visualization and analysis) // <http://www.pointcloudviz.com/>
- [9] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. // IEEE Trans. Pattern Anal. Mach. Intell, 1992, 14(2), pp. 239-256

*Евгений Олегович Кривошеин — студент Новосибирского государственного университета;
e-mail: evgeny.krivoshein@gmail.com.*

*Николай Валерьевич Снытников — к.ф.-м.н., науч.сотр. Института вычислительной математики и
математической геофизики СО РАН;
e-mail: nik@ssd.sscs.ru.*

Дата поступления — 31 мая 2017 г.