Параллельные вычислительные технологии (ПаВТ) 2016

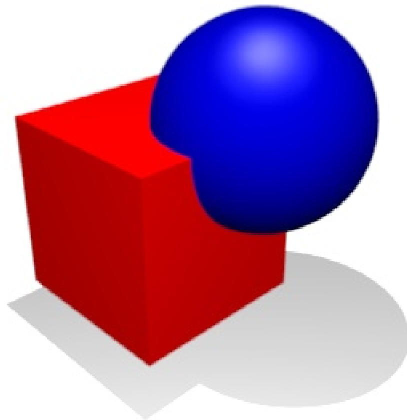
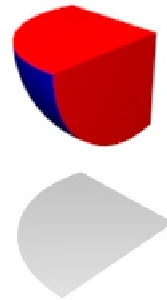
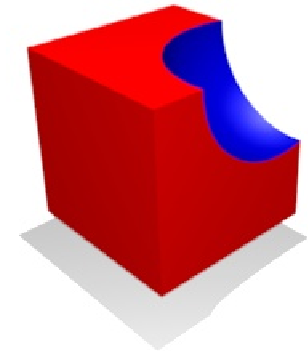
Оптимизация CSG деревьев для трассировки лучей реального времени на GPU

D.Y. Ulyanov[1,2], D.K. Bogolepov[2], V.E. Turlapov[1]
University of Nizhniy Novgorod[1], OpenCASCADE[2]

Докладчик: Турлапов В.Е.

Конструктивная блочная геометрия (*CSG*)

- *CSG* – технология моделирования твердых тел, которая строит сложные объекты путем применения булевых операций к базовым примитивам (объединение, пересечение, разность)

- Один из основных способов моделирования в 3D графике и САПР
- Алгоритмы визуализации требуют *большого объема вычислений*

Визуализация CSG моделей...

- Два базовых подхода:
  - *Вычисление границы CSG объекта*. Полученная граница тесселируется и отображается средствами 3D графики
    - Высокая трудоемкость, ограничены статическими сценами
  - *Image-based технологии*. Строят только изображение CSG объекта, не вычисляя его границы
    - Многопроходные алгоритмы, интенсивно использующие буферы глубины и трафарета: *Goldfeather algorithm*, *Sequenced Convex Subtraction* (*SCS*)
    - Требуют преобразования CSG выражения в *нормальную форму*, которая растет экспоненциально от числа CSG операций

Визуализация CSG моделей

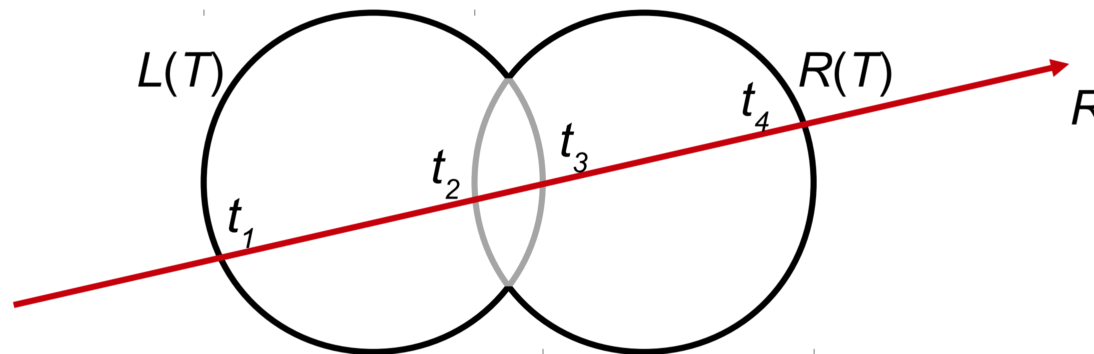
- Особенности *image-based* техник:
  - *Ограниченная сложность CSG объектов*. Интерактивная производительность для сцен средней сложности (тысячи CSG примитивов)
  - *Ограниченная масштабируемость*. Зависит от пропускной способности памяти (bandwidth-bounded)
    - Пропускная способность памяти растет незначительно
- Альтернатива – *трассировка лучей* (*ray-tracing*)
  - Существующие реализации, основанные на разбиении луча на интервалы, неэффективны и не адаптированы для GPU

Цели настоящей работы

- Разработать эффективное решение для визуализации CSG деревьев в реальном времени на GPU:
  - *Трассировка лучей как базовый алгоритм*. Адаптация для архитектуры графического процессора
  - *Оптимизация CSG деревьев*. Преобразование деревьев в форму, оптимальную для трассировки лучей
  - *Реализация кросс-платформенности*. Поддержка GPUs различных производителей, включая решения начального уровня

Kensler´s CSG ray-tracing...

- Использует информацию только о *ближайшем соударении*:
  - *T* – CSG дерево, *L*(*T*) и *R*(*T*) – левое и правое поддерево, *R* – луч



  - Луч *R* пересекается с поддеревьями *L*(*T*) и *R*(*T*): *входит в объект*, *покидает объект*, *не пересекает объект*
  - На основе двух классификаций для *L*(*T*) и *R*(*T*) принимается решение: *вернуть точку соударения*, *пересечения нет*, *перейти к поддереву* и выполнить классификацию повторно (рекурсия)

Kensler´s CSG ray-tracing: таблицы перехода

∪	Enter $R(T)$	Exit $R(T)$	Miss $R(T)$
Enter $L(T)$	*RetLIfCloser* *RetRIfCloser*	*RetRIfCloser* *LoopL*	*RetL*
Exit $L(T)$	*RetLIfCloser* *LoopR*	*LoopLIfCloser* *LoopRIfCloser*	*RetL*
Miss $L(T)$	*RetR*	*RetR*	*Miss*

∩	Enter $R(T)$	Exit $R(T)$	Miss $R(T)$
Enter $L(T)$	*LoopLIfCloser* *LoopRIfCloser*	*RetLIfCloser* *LoopR*	*Miss*
Exit $L(T)$	*RetRIfCloser* *LoopL*	*RetLIfCloser* *RetRIfCloser*	*Miss*
Miss $L(T)$	*Miss*	*Miss*	*Miss*

\	Enter $R(T)$	Exit $R(T)$	Miss $R(T)$
Enter $L(T)$	*RetLIfCloser* *LoopR*	*LoopLIfCloser* *LoopRIfCloser*	*RetL*
Exit $L(T)$	*RetLIfCloser* *RetRIfCloser* *FlipR*	*RetRIfCloser* *FlipR* *LoopL*	*RetL*
Miss $L(T)$	*Miss*	*Miss*	*Miss*

Kensler´s CSG ray-tracing: алгоритм...

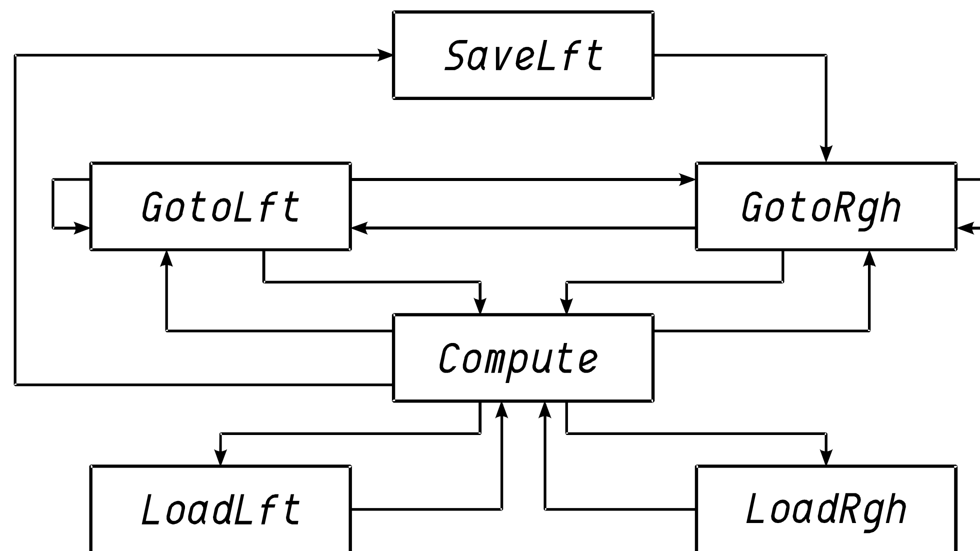
```
function Intersect(node, min)
  min_L ← min
  min_R ← min
  (t_L,N_L) ← Intersect(L(node), min_L)
  (t_R,N_R) ← Intersect(R(node), min_R)
  state_L ← Classify(t_L, N_L)
  state_R ← Classify(t_R, N_R)
  while true do
    actions ← table[state_L, state_R] // таблицы перехода
    if Miss ∈ actions then
      return miss
    if RetL ∈ actions or (RetLIfCloser ∈ actions and t_L ≤ t_R) then
      return (t_L, N_L)
    if RetR ∈ actions or (RetRIfCloser ∈ actions and t_R ≤ t_L) then
      if FlipR ∈ actions then
        N_R ← −N_R
      return (t_R, N_R)
    ...
```

Kensler´s CSG ray-tracing: алгоритм

```
...

if LoopL ∈ actions or (LoopLIfCloser ∈ actions and t_L ≤ t_R) then
  min_L ← t_L
  (t_L, N_L) ← Intersect(L(node), min_L)
  state_L ← Classify(t_L, N_L)
else
  if LoopR ∈ actions or (LoopRIfCloser ∈ actions and t_R ≤ t_L) then
    minR ← t_R
    (t_R, N_R) ← Intersect(R(node), min_R)
    stateR ← Classify(t_R, N_R)
  else
    return miss
```

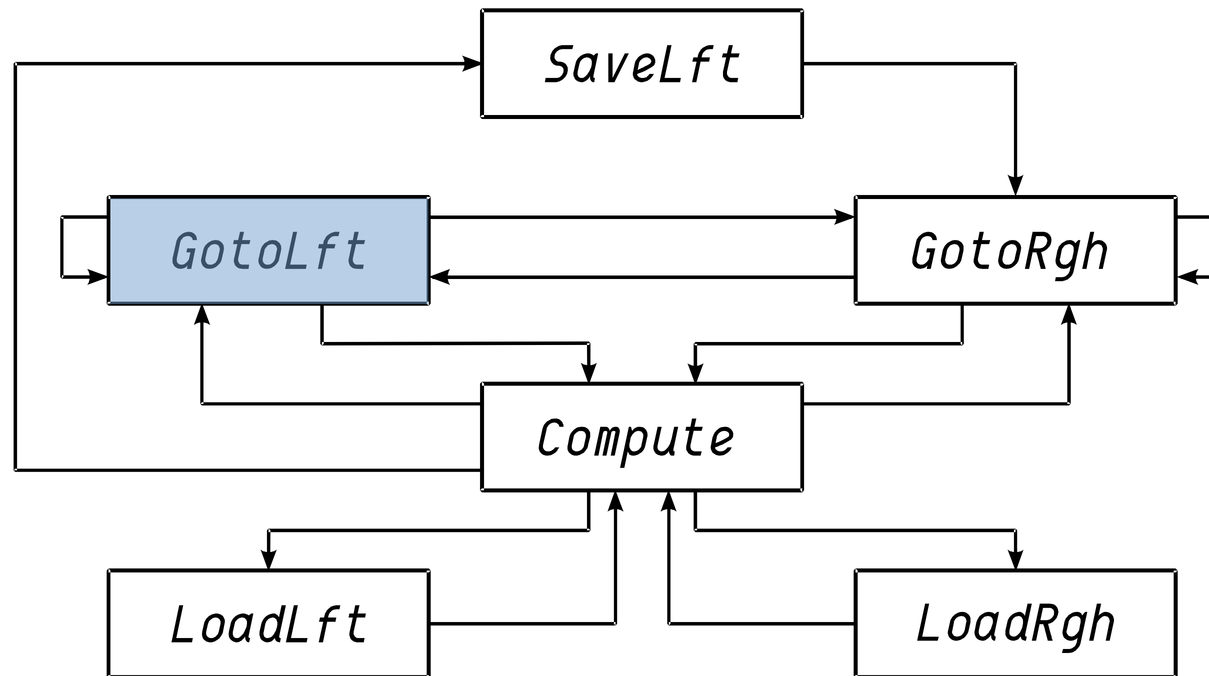
Итеративный CSG ray-tracing на стеке...

- Идея: вводится высокоуровневый *автомат с памятью*



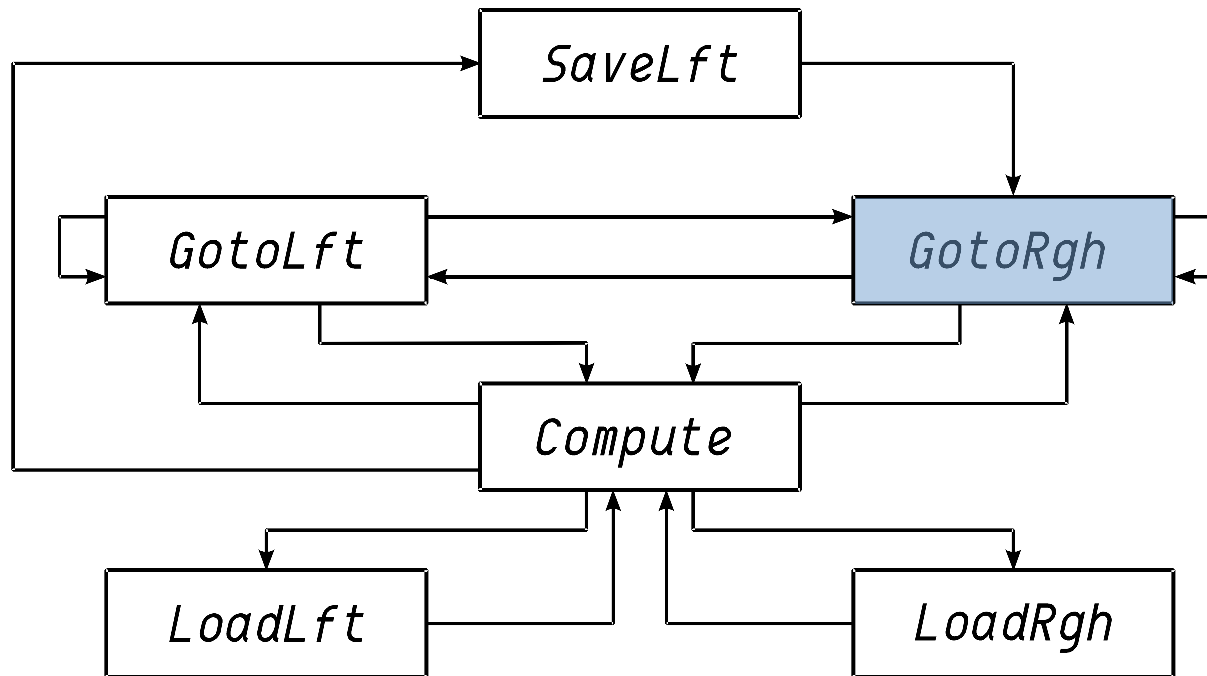
- Типы состояний: *вычисление пересечений* с поддеревьями $L(T)$ и $R(T)$ и *применение таблиц перехода* для обработки (классификации) точек соударения

Итеративный CSG ray-tracing на стеке...



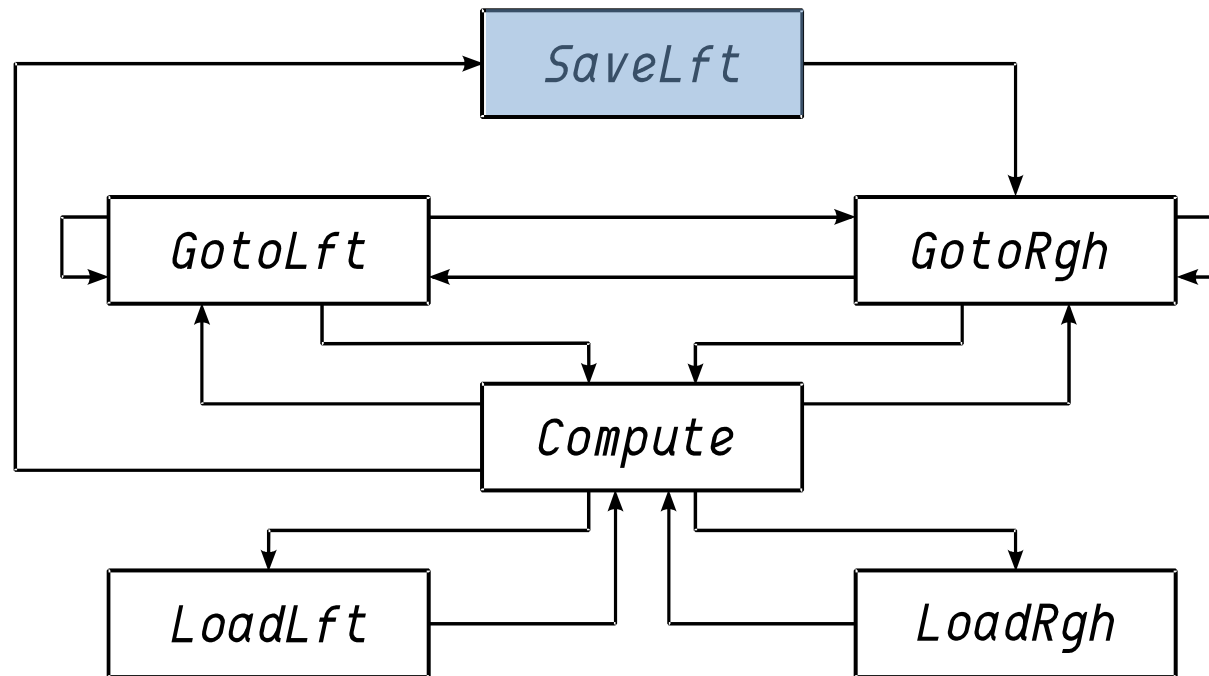
GotoLft – вычисление пересечения с поддеревом *L*(*T*)

Итеративный CSG ray-tracing на стеке...



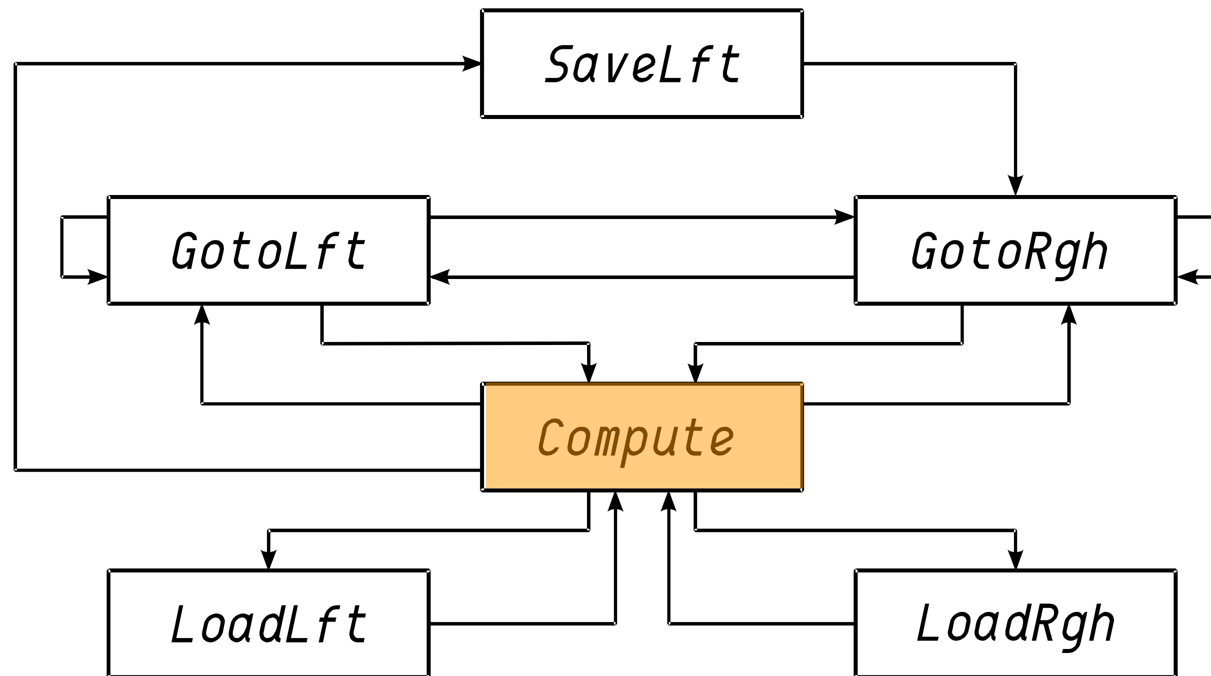
GotoRgh – вычисление пересечения с поддеревом *R*(*T*)

Итеративный CSG ray-tracing на стеке...



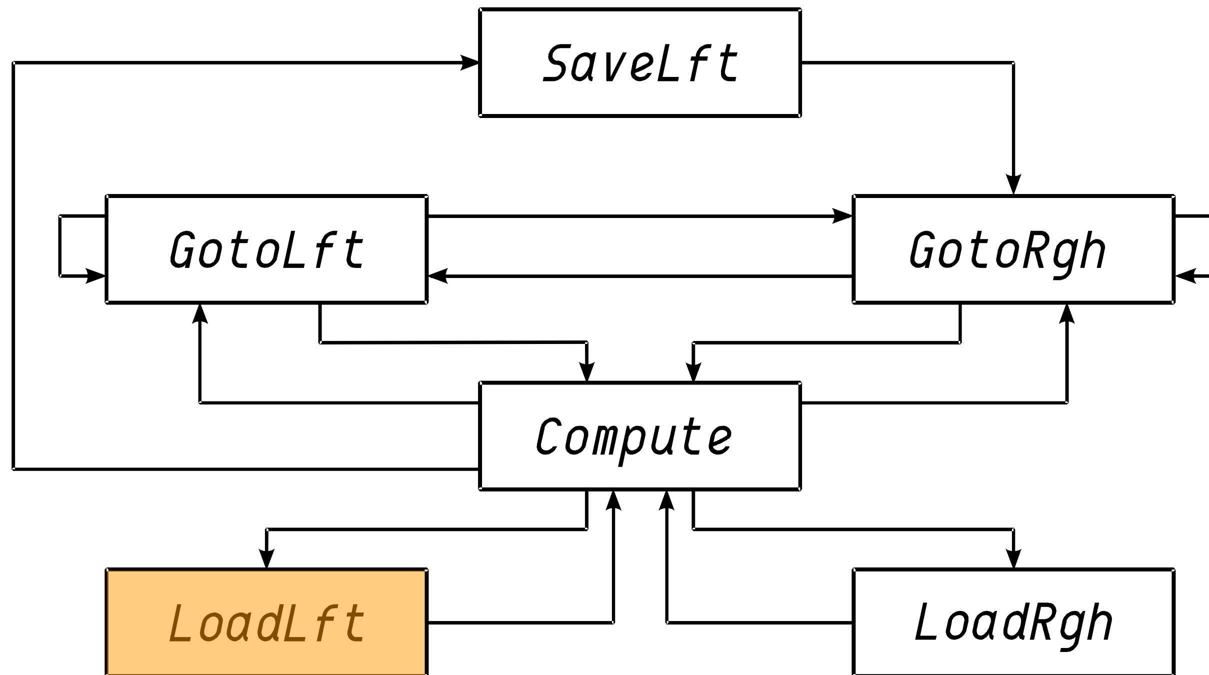
SaveLft – сохранение точки пересечения с поддеревом $L(T)$ и переход к состоянию *GotoRgh*

Итеративный CSG ray-tracing на стеке...



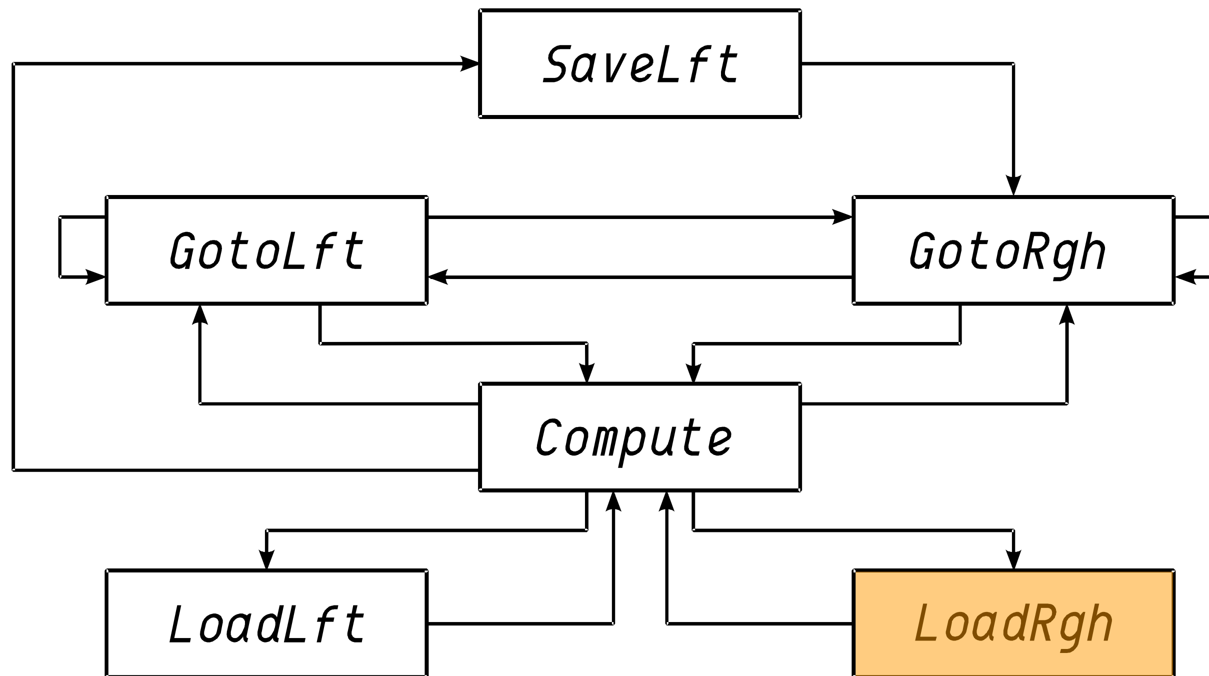
Compute – применение таблиц перехода (алгоритм Kensler´a)

Итеративный CSG ray-tracing на стеке...



LoadLft – загрузка точки пересечения с *L*(*T*)
и переход к состоянию *Compute*

Итеративный CSG ray-tracing на стеке



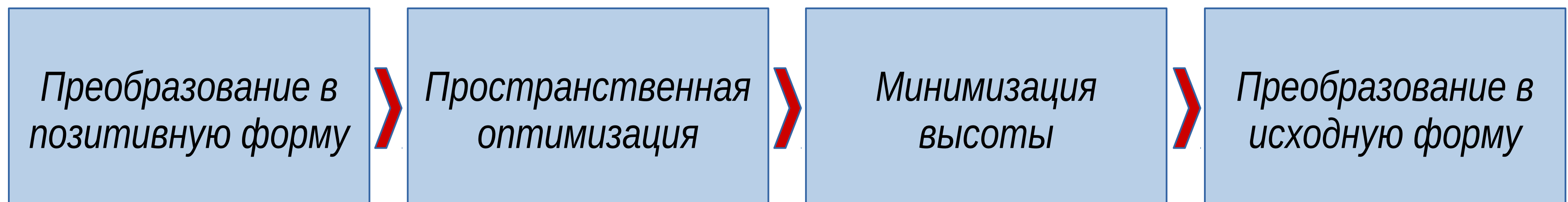
LoadRgh – загрузка точки пересечения с *R*(*T*)
и переход к состоянию *Compute*

Итеративный CSG ray-tracing: алгоритм

```
tmin ← 0
node ← V  // виртуальный корень: имеет CSG дерево в левом поддереве

(t_L, N_L) ← invalid
(t_R, N_R) ← invalid

PushAction(Compute) // сохраняем в стек операцию
action ← GotoLft

while true do
 if action ≡ SaveLft then
  tmin ← PopTime()
  PushPrimitive(t_L, N_L) // сохраняем точку пересечения в стеке
  action ← GotoRgh
 if action ∈ {GotoLft, GotoRgh} then
  GoTo()
 if action ∈ {LoadLft, LoadRgh, Compute} then
  Compute()
```

Оптимизация CSG деревьев...

- Производительность существенно зависит от топологии CSG дерева, которое зачастую формируется автоматизированными инструментами
- Задача: преобразовать исходное дерево *T* в эквивалентное дерево *T´*, которое будет *близко к сбалансированному* и иметь *близкое число узлов*
- Предложена последовательность алгоритмов оптимизации:



Преобразование в позитивную форму

- В *позитивной форме* CSG выражение представляется только *коммутативными* операциями – $\cup$ и $\cap$
- Преобразование выполняется путем обхода дерева в *прямом порядке*:

$$\overline{x \cup y} = \overline{x} \cap \overline{y}$$

$$\overline{x \cap y} = \overline{x} \cup \overline{y}$$

$$x - y = x \cap \overline{y}$$

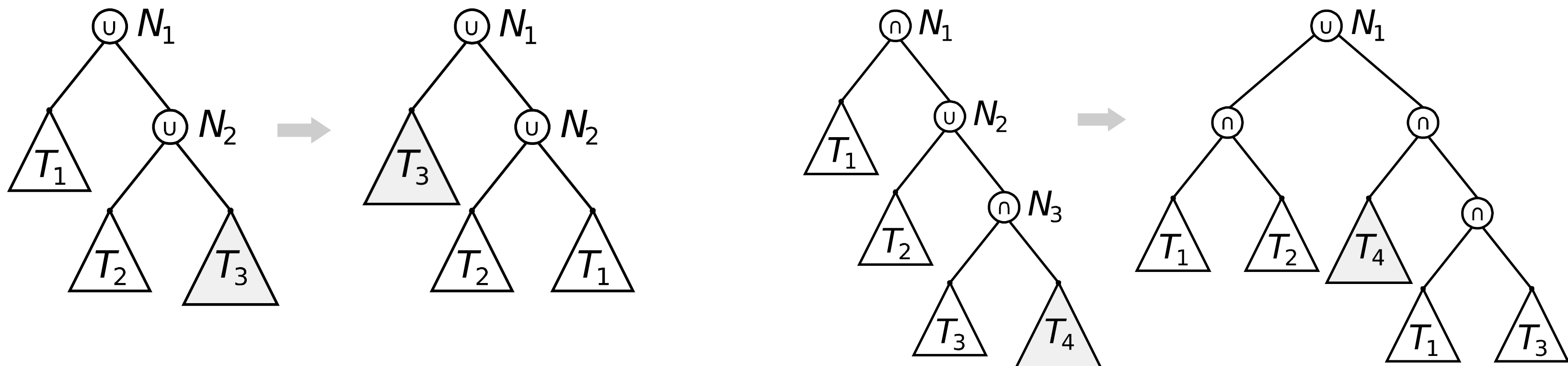
- Преобразование в исходную форму выполняется с помощью обход дерева в *обратном порядке*
- *Дочерние узлы любого узла T´ (в позитивной форме) можно менять местами*

Пространственная оптимизация

- Обеспечивает *минимизацию ограничивающих объемов* (AABB) узлов дерева *T´* (ускоряет ранний выход)
- Процедура основана на понятии *treelet´*а – произвольный узел дерева с набором непосредственных потомков
  - Листья *treelet´*а могут не соответствовать листьям дерева!
- Идея: последовательно выбирать *treelet´*ы, содержащие узлы с одинаковой булевой операцией ($\cup$ или $\cap$) и реструктурировать их с целью повышения пространственной когерентности
- Каждый *treelet* перестраивается по принципам BVH дерева (SAH based binned builder), обеспечивая плотную упаковку узлов

Минимизация высоты CSG дерева

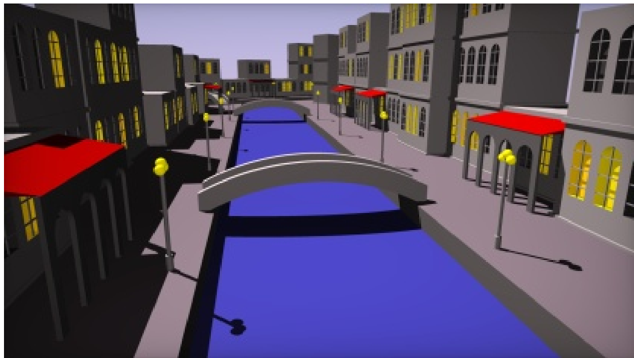
- Обеспечивает минимизацию высоты дерева *T´* (балансирует дерево), уменьшая требования к размеру стека
- Для краткости, назовем дочерний узел с *большей* высотой *тяжелым* дочерним узлом



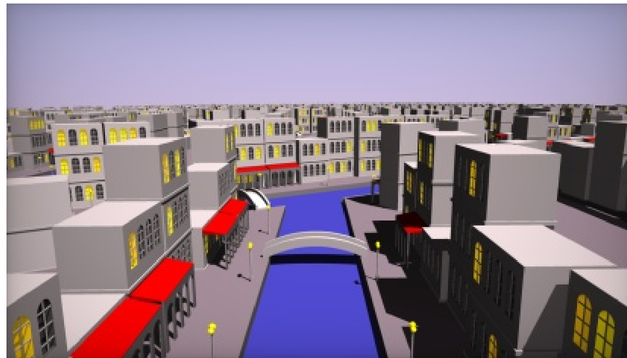
- Идея: выполнять преобразования, аналогичные поворотам, поднимая тяжёлые узлы с целью уменьшения высоты *treelet´*ов

Результаты...

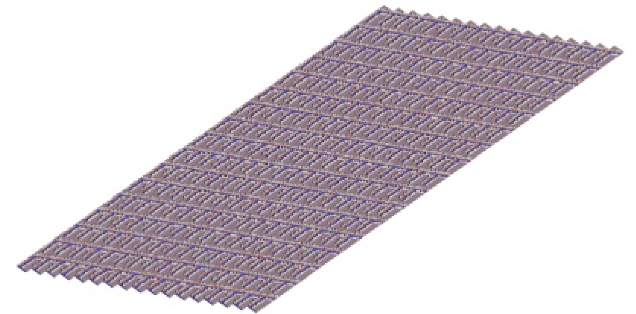
3385 CSG primitives



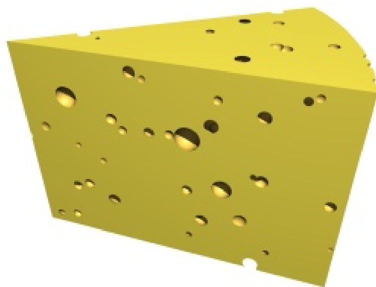
343K CSG primitives



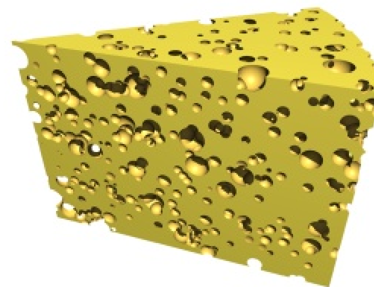
987K CSG primitives



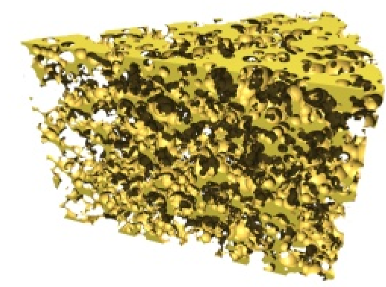
1000 CSG primitives



8000 CSG primitives



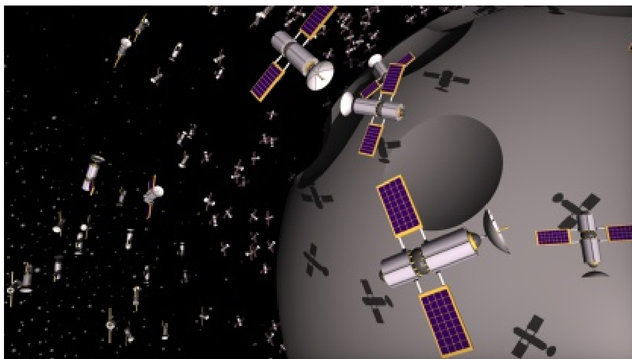
32000 CSG primitives



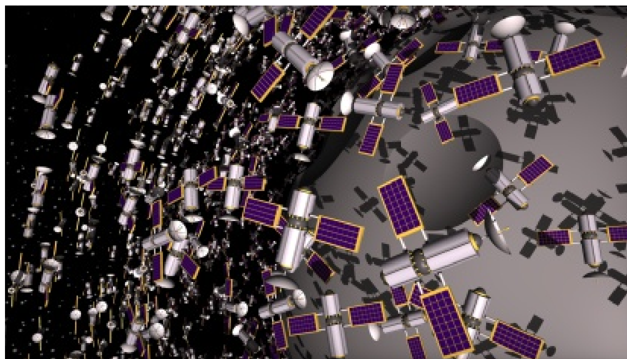
Результаты...

Пример динамической сцены:
все спутники движутся по уникальным орбитам

87 565 CSG primitives



1 120 065 CSG primitives



1 120 065 CSG primitives

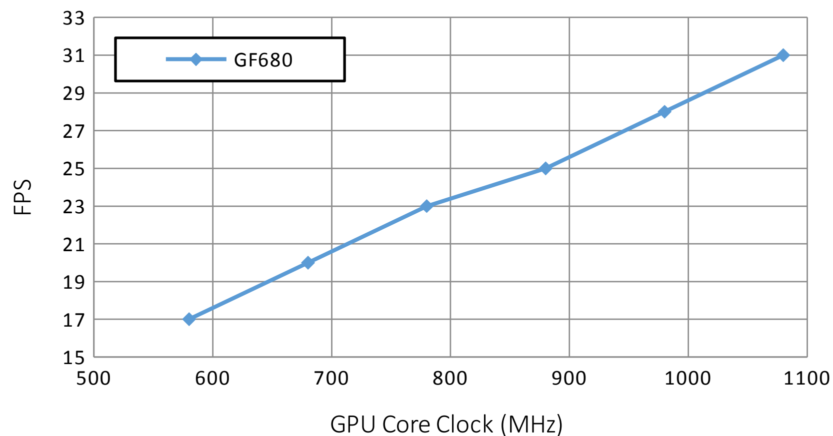
Результаты...

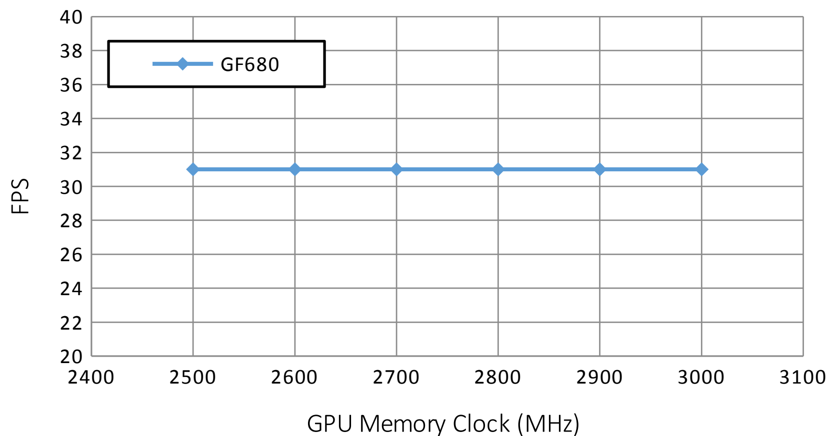
«–» – без пространственной оптимизации, «+» – с оптимизацией

Scene	Primitives	Tree Depth	Intel 4000		Radeon HD 7870		GeForce GTX 680	
			–	+	–	+	–	+
City (a)	3385	14	7	7.5	50	60	51	57
City (b)	343589	22	1.8	4.5	6.5	17	8	22
City (c)	987218	24	**2.3**	**7**	**6.7**	**18**	**8.3**	**21**
Cheese (a)	1002	11	0.4	17	4.6	110	5.8	128
Cheese (b)	8002	14	N/A	6.5	0.5	28	0.5	32
Cheese (c)	32002	17	**N/A**	**0.5**	**N/A**	**3.7**	**N/A**	**4**
Satellites (a)	87565	7	5	9	26	67	29	65
Satellites (b)	1120065	7	2.8	4.5	8	18	7	15
Satellites (c)	1120065	7	**2.5**	**4.5**	**4.2**	**9**	**5.6**	**12**

Результаты...

- Проверка масштабируемости решения





Вывод: линейная масштабируемость по частоте GPU и отсутствие чувствительности к частоте памяти

Результаты

- Решение обеспечивает интерактивную визуализацию CSG сцен, содержащих свыше 1 миллиона примитивов (поверхности второго порядка)
  - Высокая производительность на интегрированных GPU
  - Алгоритм ограничен вычислительными ресурсами, а не пропускной способностью памяти (*computational-bound*)
- Предложенный алгоритм оптимизации CSG деревьев позволяет поднять производительность до 3-х раз и более (на сложных сценах)
- Алгоритм основан на трассировке лучей и легко расширяется для поддержки различных визуальных эффектов (прозрачность, тени, отражения, преломления или глобальное освещение)