刚体动力学
并行编程与实践

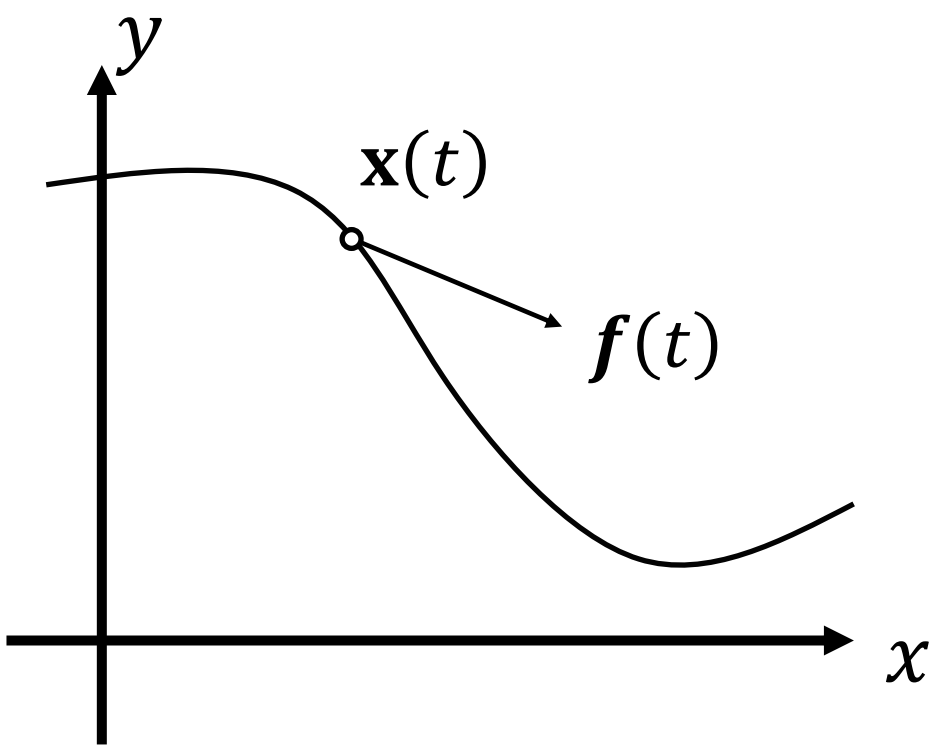
何小伟
中国科学院软件研究所
2023.4.16

大纲

- 刚体动力学基础
- 宽阶段碰撞检测
- 窄阶段碰撞检测
- 刚体动力学求解
- 场景演示

刚体动力学基础

- 质点
  - 具有一定质量
  - 不存在体积或者形状
  - 不考虑旋转
  - 描述一个质点状态的物理量包括
    - 质量：m
    - 位置：$\mathbf{x}(t)$
    - 速度：$\mathbf{v}(t) = \dot{\mathbf{x}}(t)$
    - 加速度：$\mathbf{a}(t) = \dot{\mathbf{v}}(t)$
    - 外力：$\mathbf{f}(t)$

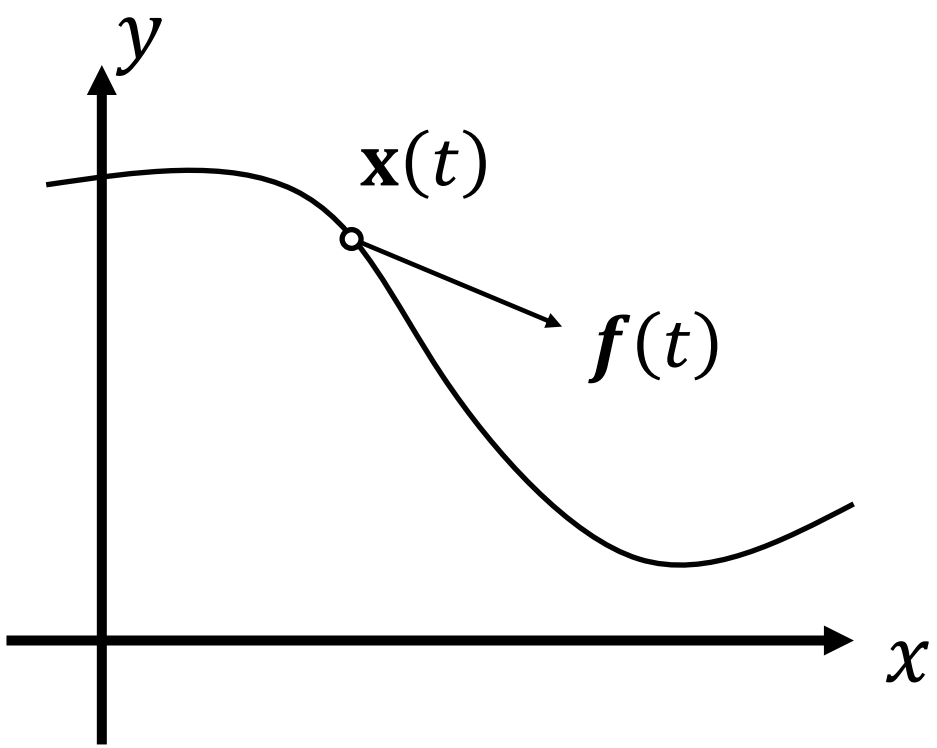


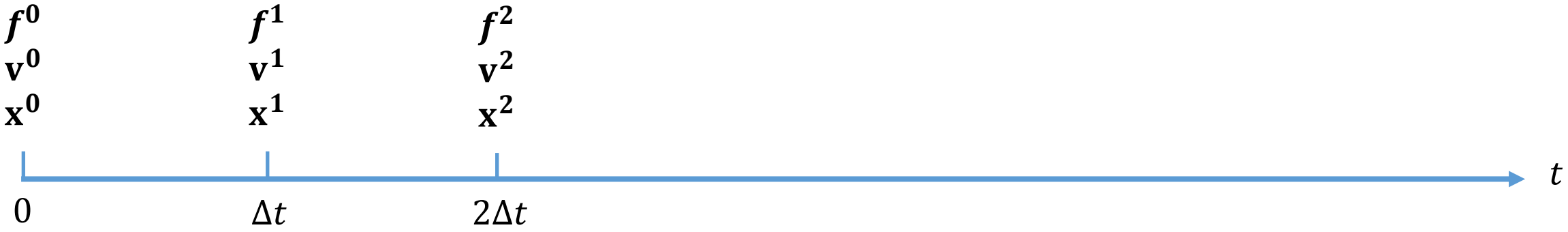
刚体动力学基础

- 质点运动状态更新
  - 牛顿第二定律

$$\mathbf{v} \longleftarrow \mathbf{v} + \Delta t m^{-1}\mathbf{f}$$

$$\mathbf{x} \longleftarrow \mathbf{x} + \Delta t \mathbf{v}$$





刚体动力学基础

- 刚体状态变量
  - 具有一定质量
  - 存在体积或者形状
  - 考虑旋转
  - 描述一个质点状态的物理量包括

平移

- 质量$\mathbf{M}$
- 重心坐标$\mathbf{x}(t)$
- 平移速度$\mathbf{v}(t) = \dot{\mathbf{x}}(t)$
- 平移加速度$\mathbf{a}(t) = \dot{\mathbf{v}}(t)$
- 外力:$\mathbf{f}(t)$

旋转

- 惯性张量：$\mathbf{I}$
- 旋转矩阵：$\mathbf{R}(t)$ or $\mathbf{q}(t)$
- 角速度：$\boldsymbol{\omega}(t)$
- 角加速度：$\dot{\mathbf{q}}(t)$
- 扭矩：$\boldsymbol{\tau}(t)$

刚体动力学基础

- 刚体运动状态更新

平移

$$\mathbf{v} \longleftarrow \mathbf{v} + \Delta t m^{-1}\mathbf{f}$$

$$\mathbf{x} \longleftarrow \mathbf{x} + \Delta t \mathbf{v}$$

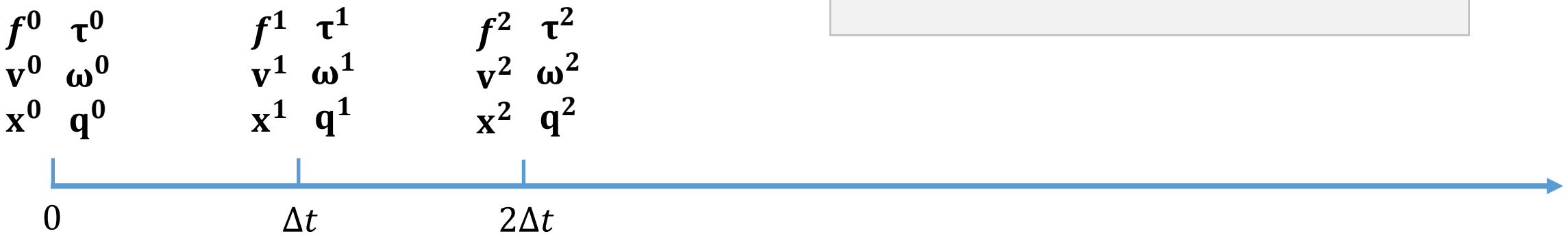
旋转

$$\mathbf{R} \longleftarrow \mathbf{R}(\mathbf{q})$$

$$\mathbf{I} \longleftarrow \mathbf{R}\mathbf{I}_{\mathbf{ref}}\mathbf{R}^{\mathrm{T}}$$

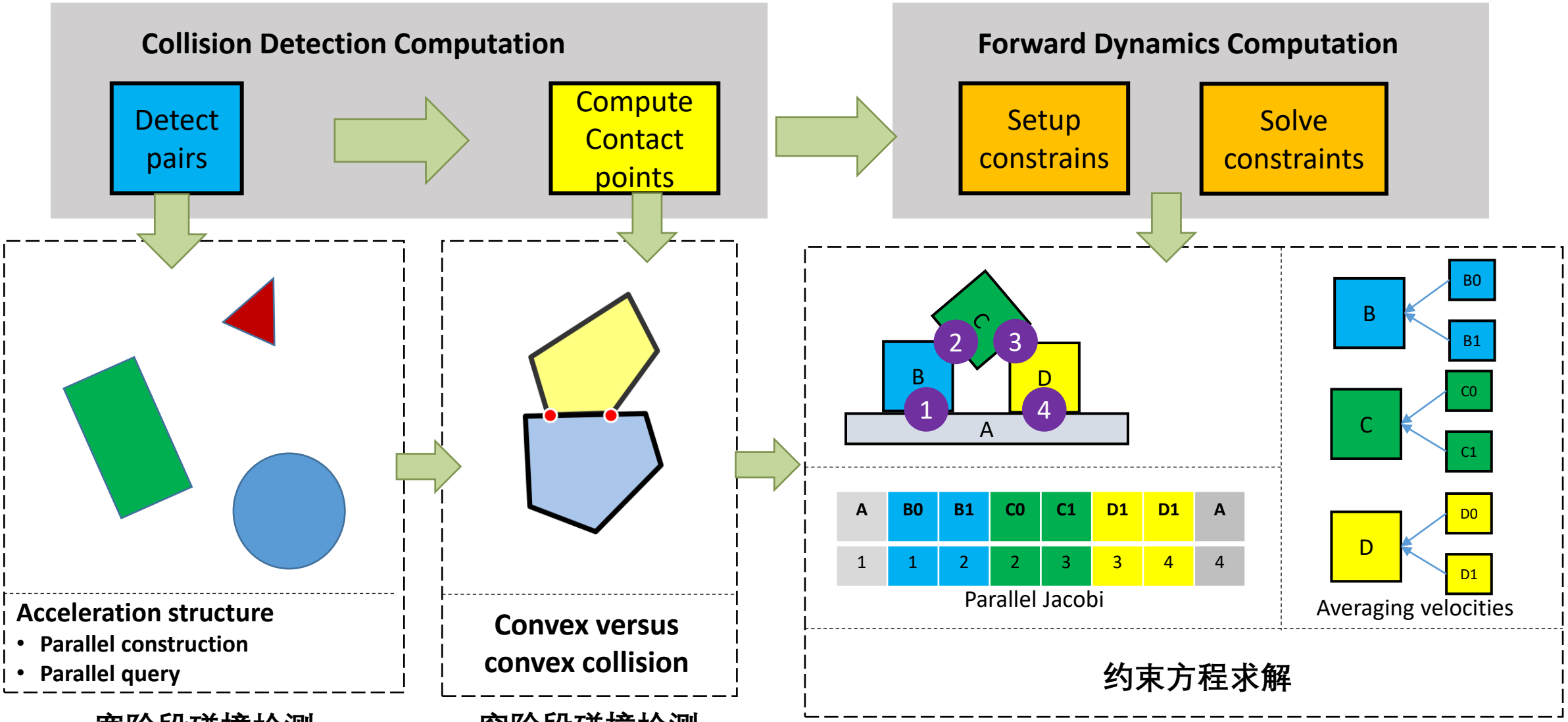
$$\boldsymbol{\omega} \longleftarrow \boldsymbol{\omega} + \Delta t(\mathbf{I})^{-1}\boldsymbol{\tau}$$

$$\mathbf{q} \longleftarrow \mathbf{q} + \frac{\Delta t}{2}[\boldsymbol{\omega} \quad \mathbf{0}] \times \mathbf{q}$$



刚体动力学基础

- 刚体动力学仿真流程图



宽阶段碰撞检测　　窄阶段碰撞检测

宽阶段碰撞检测

- 包围盒（Bounding Volume）
- 空间划分算法（Spatial Partitioning Algorithm）
  - 排序扫描算法（ Sort and Sweep ）
  - 均匀网格（ Uniform Grid ）
  - 层次包围盒（ Bounding Volume Hierarchy ）
- Linear BVH
  - 莫顿码（Morton Code）
  - 并行构建（Parallel Construction）
  - 并行遍历（Parallel Traversal）

包围盒（Bounding Volume）

- 目标：快速剔除不可能发生的碰撞的物体，降低计算量

包围盒（Bounding Volume）

- 常用包围盒

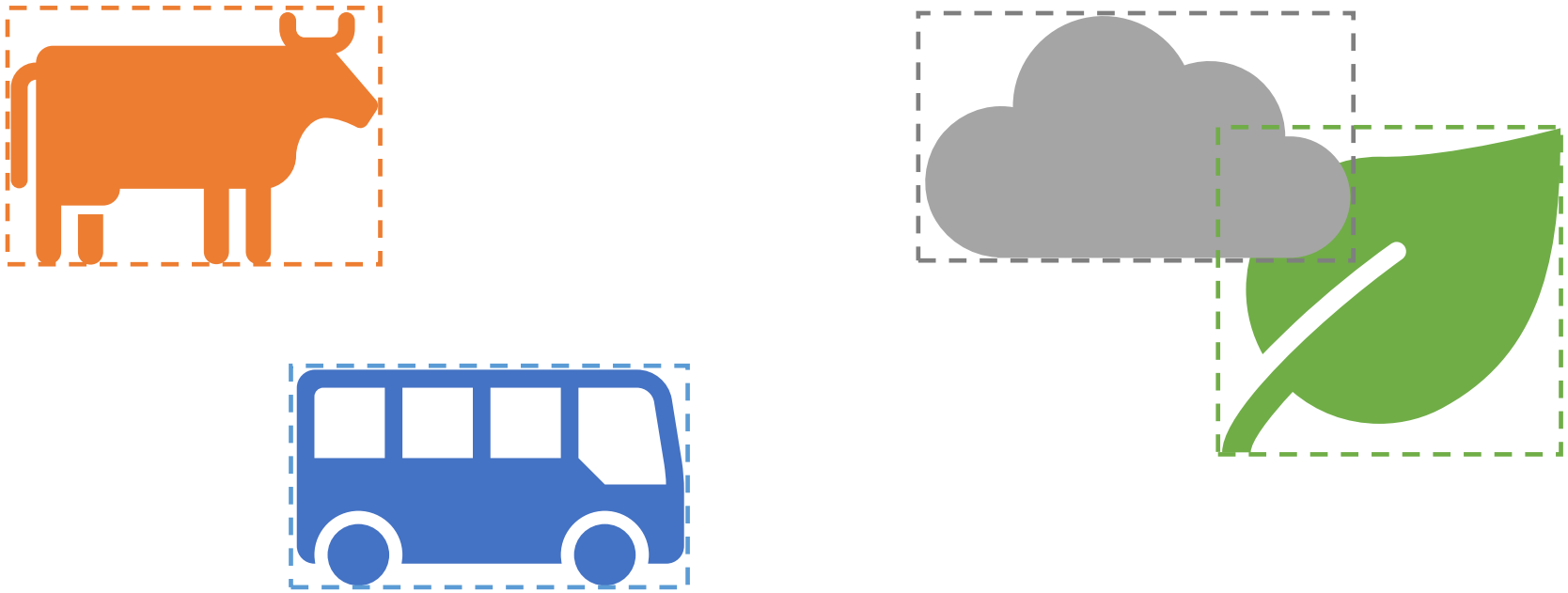
Axis-Aligned Bounding Box
(**AABB**)

Oriented Bounding Box
(**OBB**)

Bounding Sphere

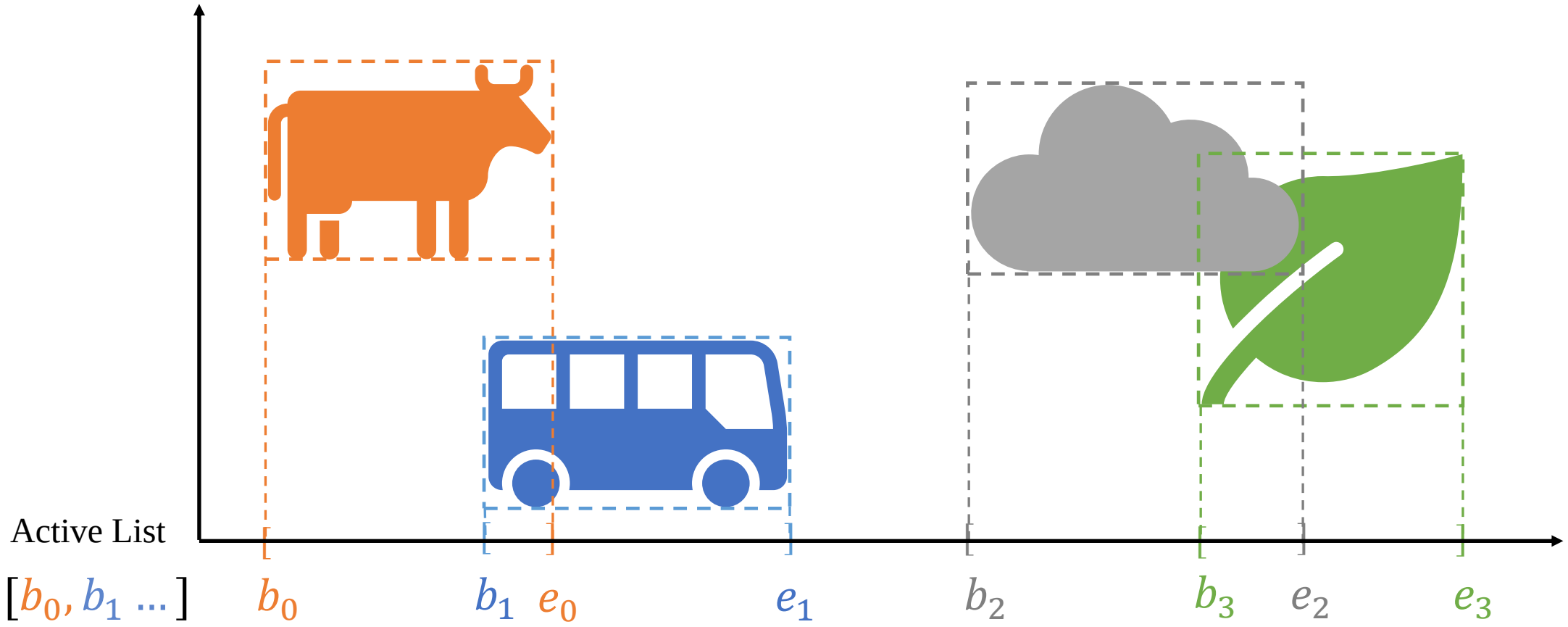
包围盒（Bounding Volume）

- AABB



空间划分算法

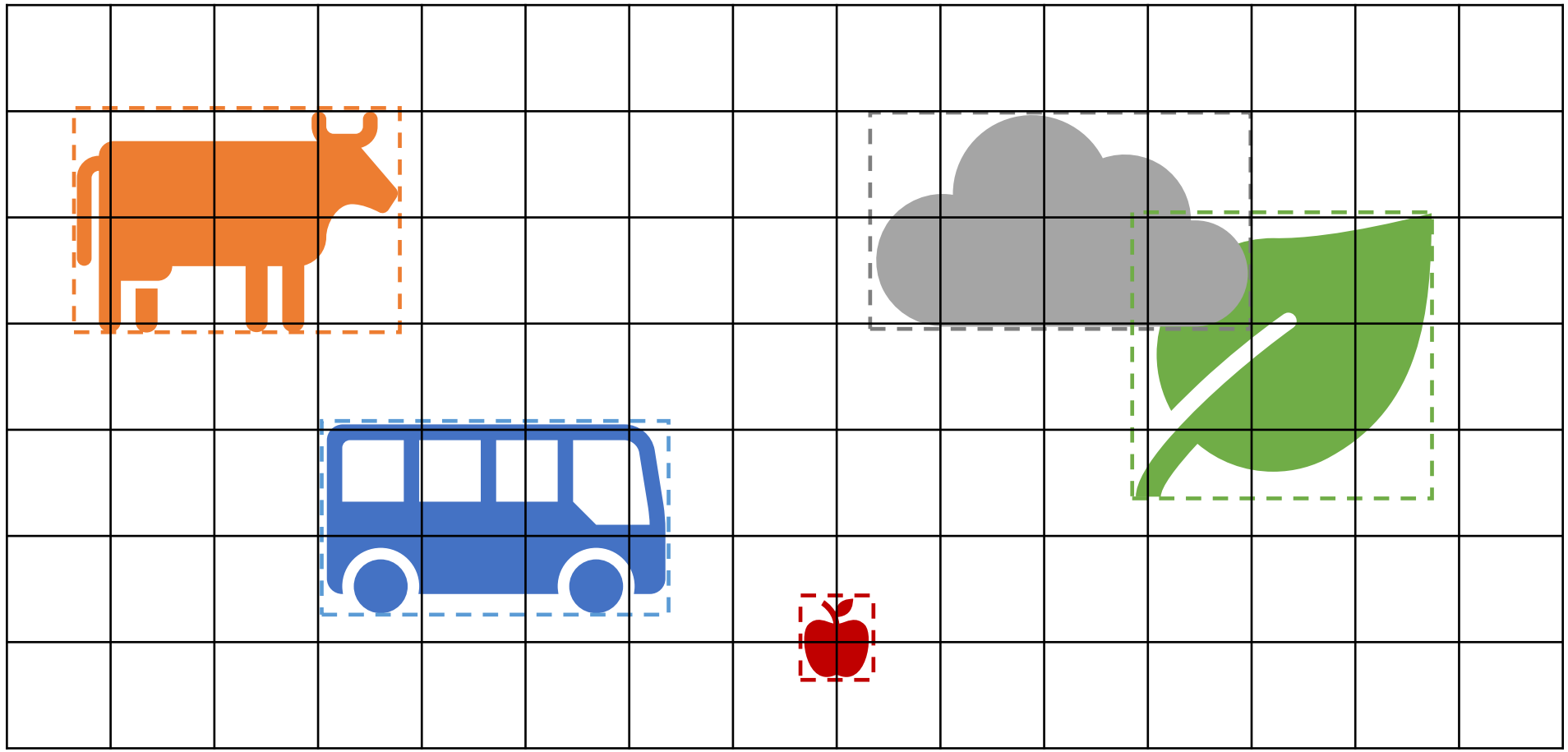
- 排序扫描算法（ Sort and Sweep ）



适合CPU实现，如Bullet引擎btAxisSweep3

空间划分算法

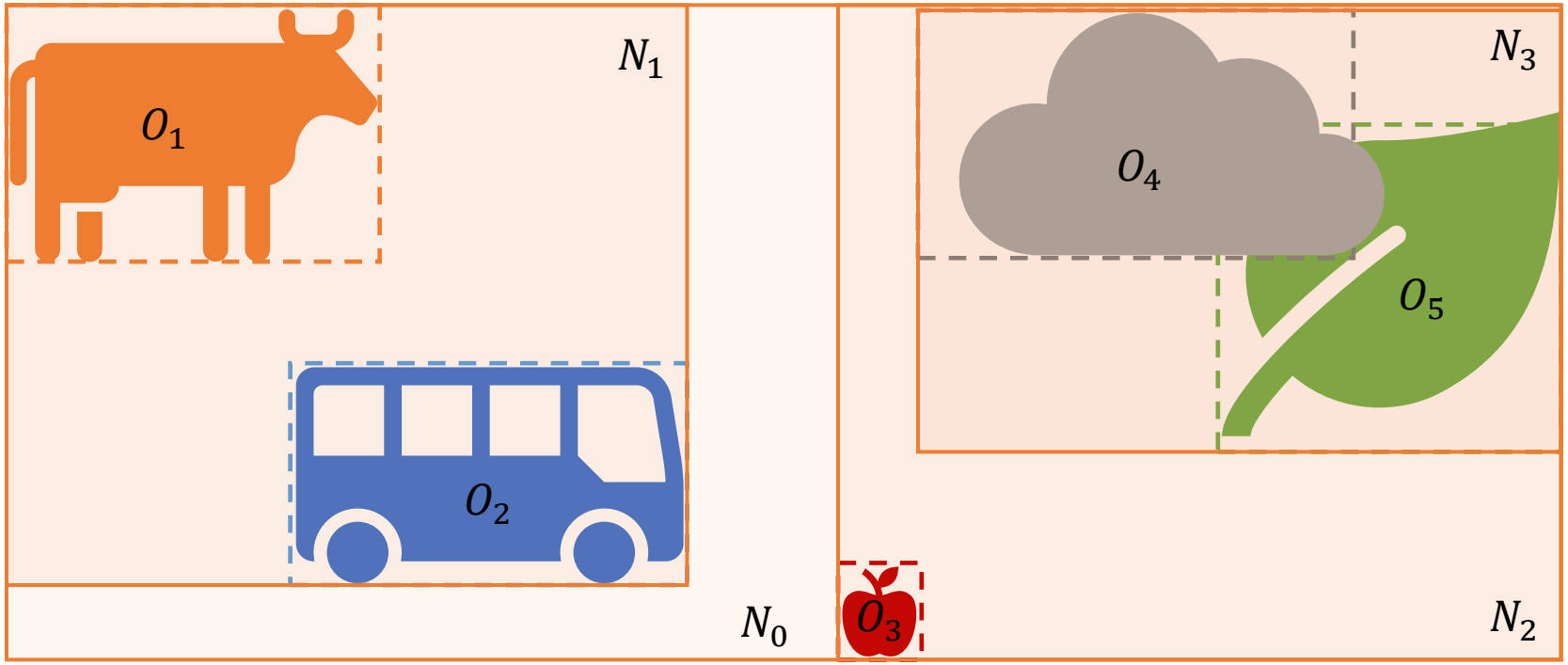
- 均匀网格（ Uniform Grid ）

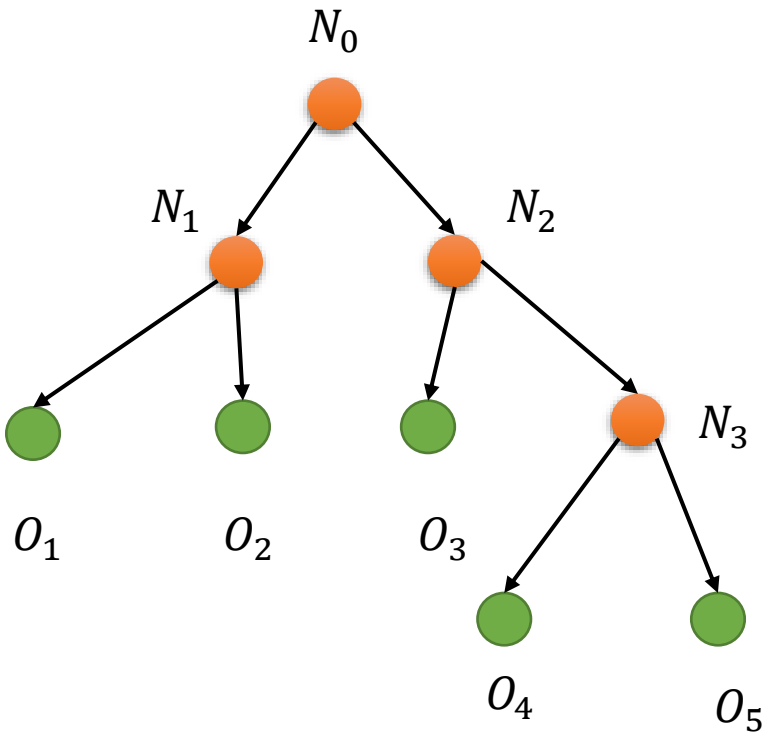


适合物体尺寸单一、分布均匀场景

空间划分算法

- 层次包围盒（ Bounding Volume Hierarchy ）





GPU如何并行构建？

Linear BVH

- 莫顿码（Morton Code）

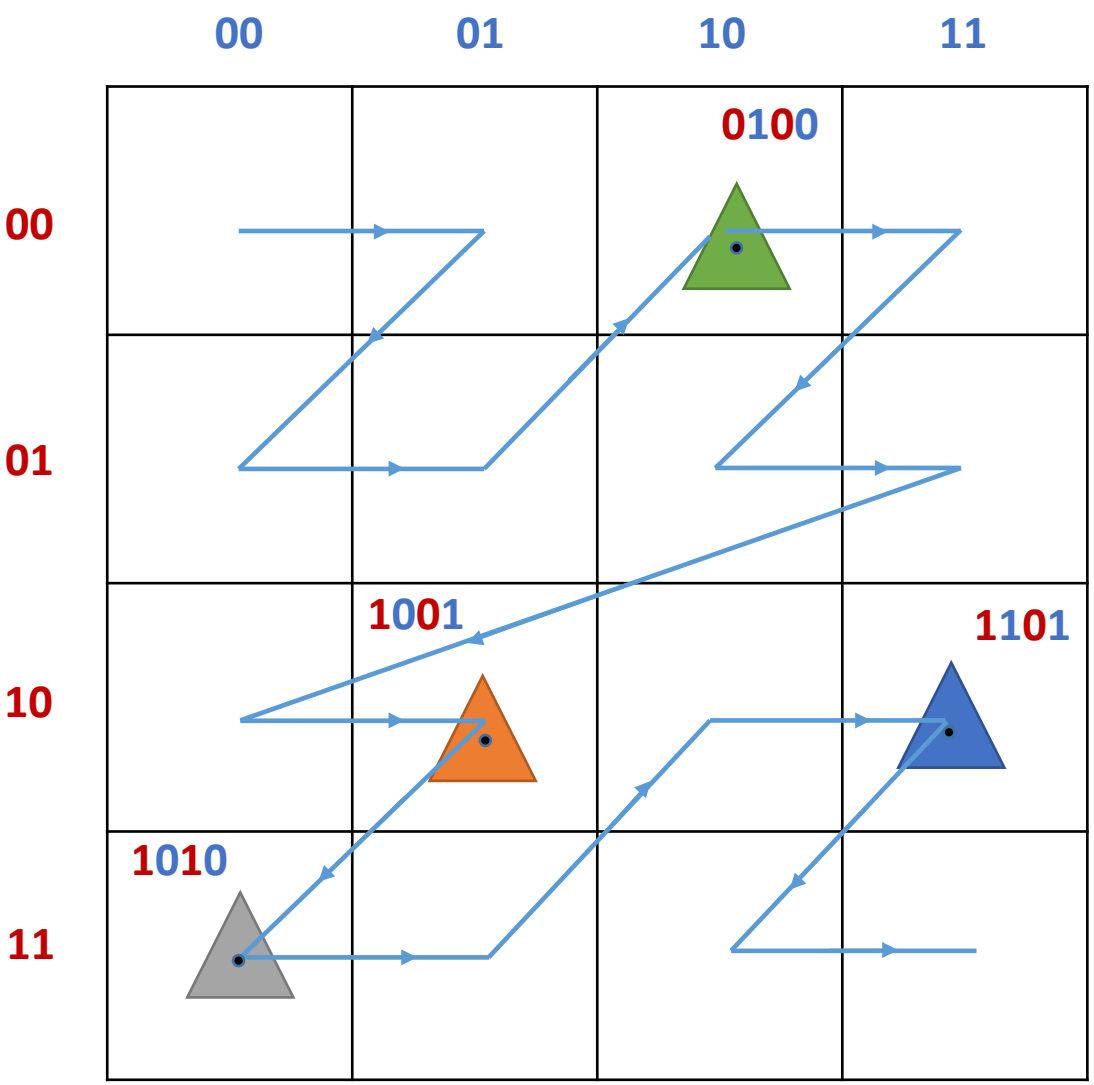
```
// Expands a 10-bit integer into 30 bits
// by inserting 2 zeros after each bit.
__device__ uint expandBits(uint v)
{
    v = (v * 0x00010001u) & 0xFF0000FFu;
    v = (v * 0x00000101u) & 0x0F00F00Fu;
    v = (v * 0x00000011u) & 0xC30C30C3u;
    v = (v * 0x00000005u) & 0x49249249u;
    return v;
}

// Calculates a 30-bit Morton code for the
// given 3D point located within the unit cube [0,1].
template<typename Real>
__device__ uint morton3D(Real x, Real y, Real z)
{
    x = min(max(x * Real(1024), Real(0)), Real(1023));
    y = min(max(y * Real(1024), Real(0)), Real(1023));
    z = min(max(z * Real(1024), Real(0)), Real(1023));
    uint xx = expandBits((uint)x);
    uint yy = expandBits((uint)y);
    uint zz = expandBits((uint)z);
    return xx * 4 + yy * 2 + zz;
}
```

	x: 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
y: 0 000	000000	000001	000100	000101	010000	010001	010100	010101
1 001	000010	000011	000110	000111	010010	010011	010110	010111
2 010	001000	001001	001100	001101	011000	011001	011100	011101
3 011	001010	001011	001110	001111	011010	011011	011110	011111
4 100	100000	100001	100100	100101	110000	110001	110100	110101
5 101	100010	100011	100110	100111	110010	110011	110110	110111
6 110	101000	101001	101100	101101	111000	111001	111100	111101
7 111	101010	101011	101110	101111	111010	111011	111110	111111

Linear BVH

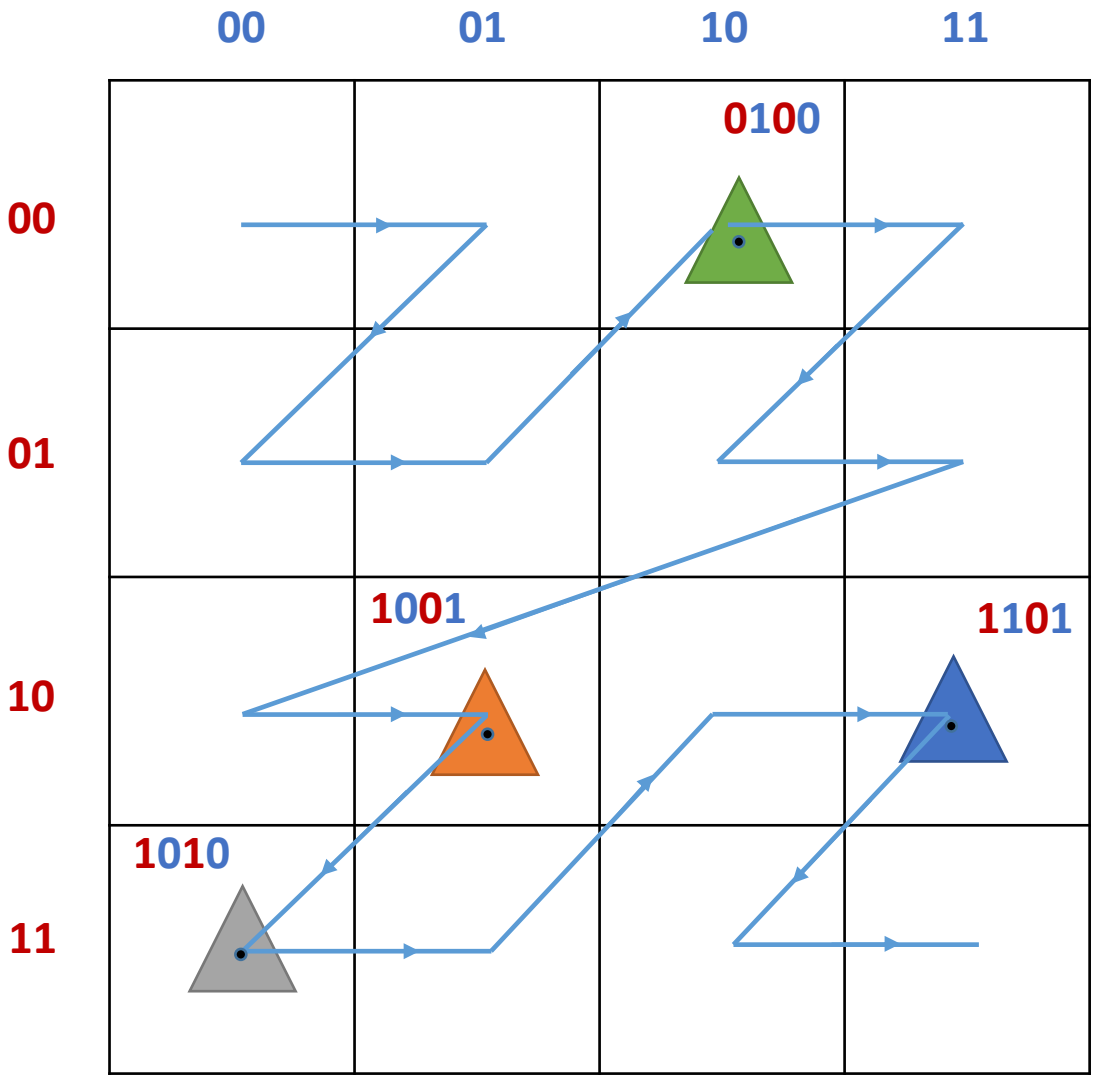
- 莫顿码（Morton Code）

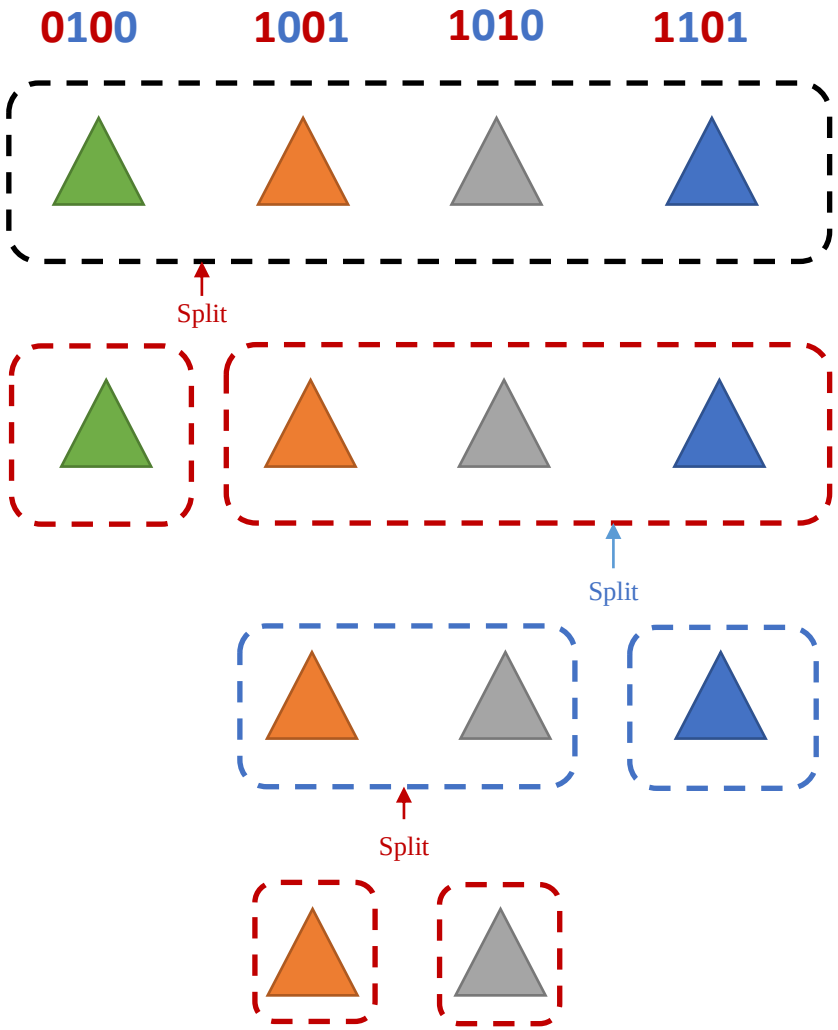
```
// Expands a 10-bit integer into 30 bits
// by inserting 2 zeros after each bit.
__device__ uint expandBits(uint v)
{
    v = (v * 0x00010001u) & 0xFF0000FFu;
    v = (v * 0x00000101u) & 0x0F00F00Fu;
    v = (v * 0x00000011u) & 0xC30C30C3u;
    v = (v * 0x00000005u) & 0x49249249u;
    return v;
}

// Calculates a 30-bit Morton code for the
// given 3D point located within the unit cube [0,1].
template<typename Real>
__device__ uint morton3D(Real x, Real y, Real z)
{
    x = min(max(x * Real(1024), Real(0)), Real(1023));
    y = min(max(y * Real(1024), Real(0)), Real(1023));
    z = min(max(z * Real(1024), Real(0)), Real(1023));
    uint xx = expandBits((uint)x);
    uint yy = expandBits((uint)y);
    uint zz = expandBits((uint)z);
    return xx * 4 + yy * 2 + zz;
}
```



Linear BVH

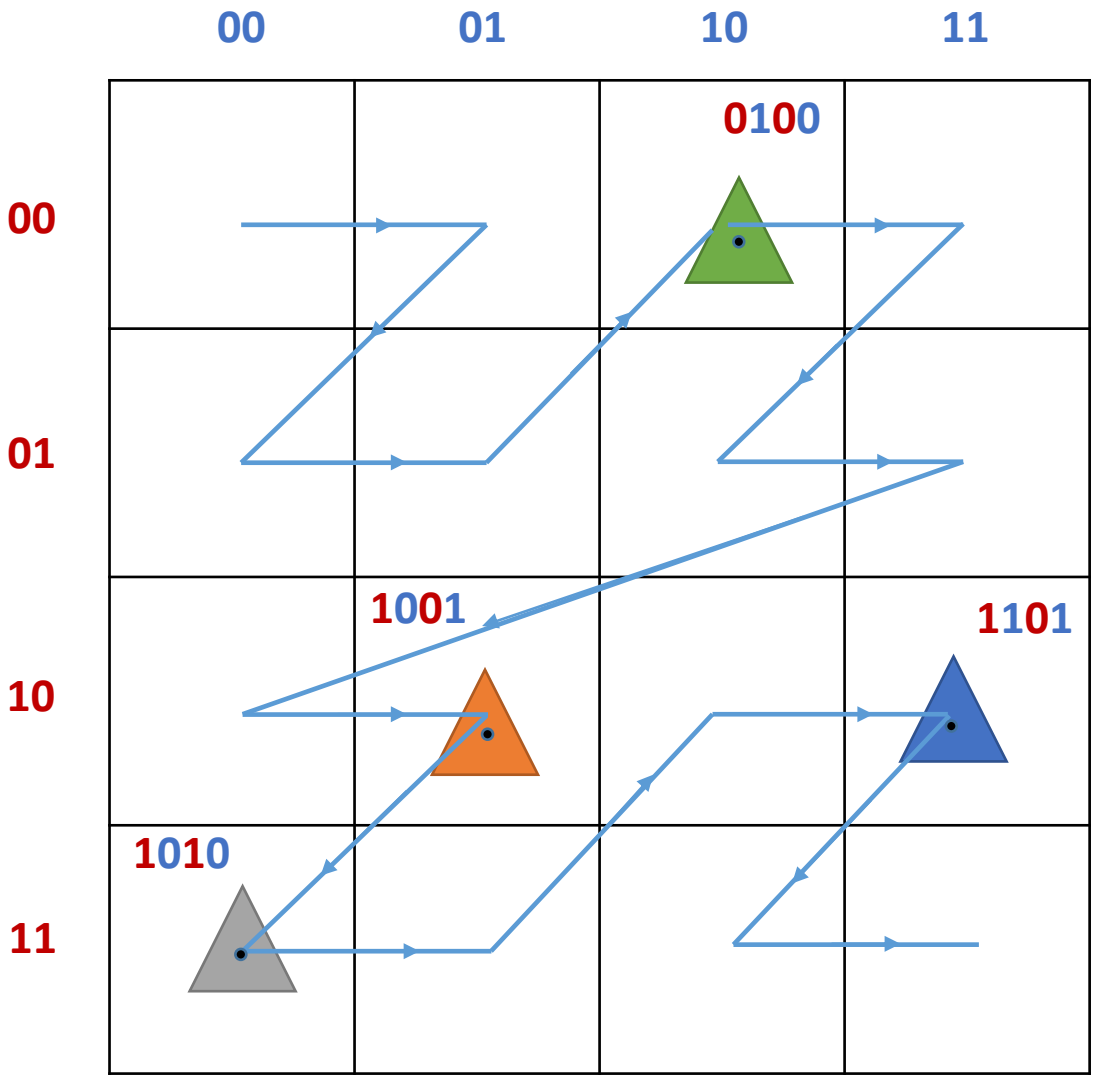
- 莫顿码（Morton Code）

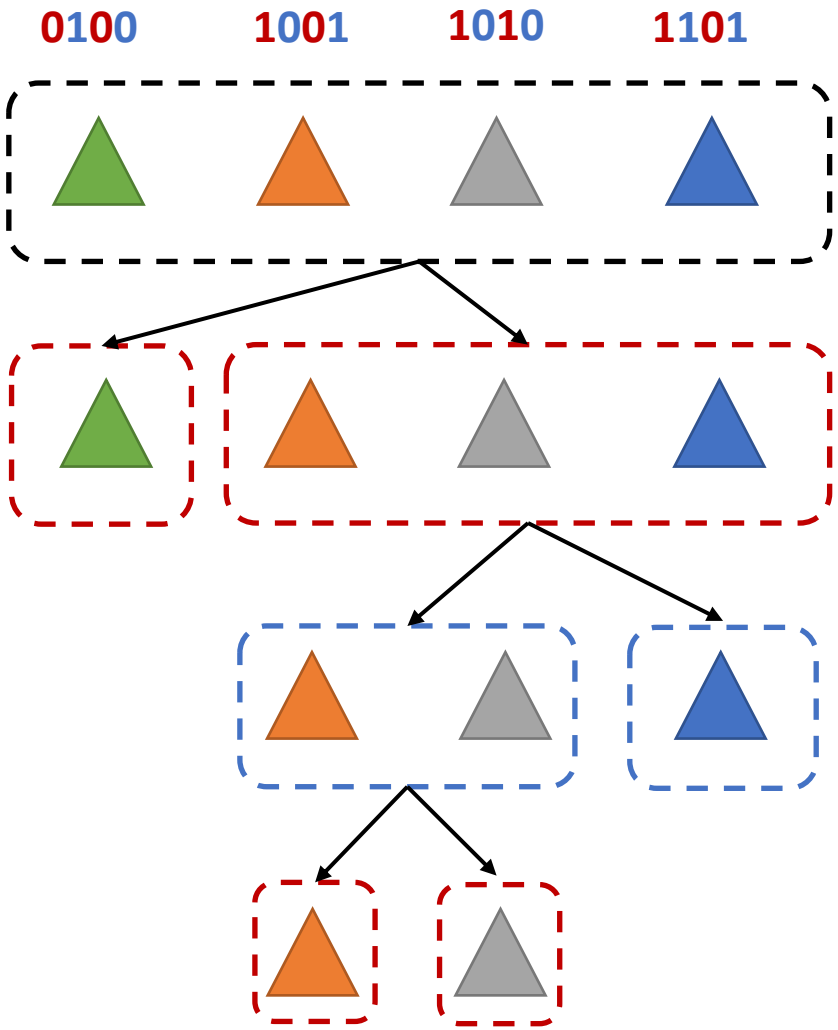




Linear BVH

- 莫顿码（Morton Code）

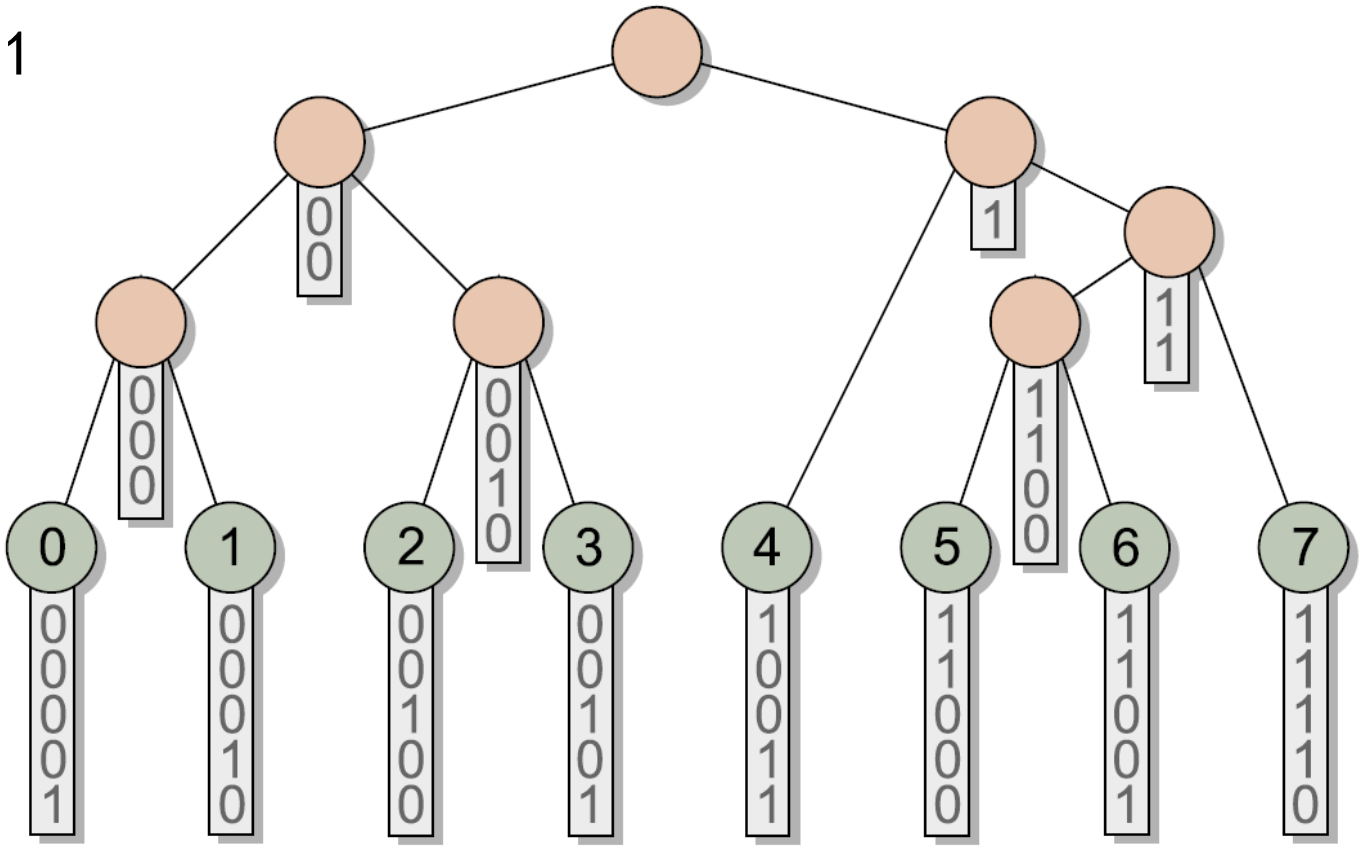




Linear BVH

- 有序二叉压缩前缀树（Ordered binary radix tree）
  - 内部节点数量=叶子节点数量−1
  - 假设$\delta(i,j)$代表叶子节点i,j的最长公共前缀，则对任意$i',j' \in [i,j]$，下述等式恒成立

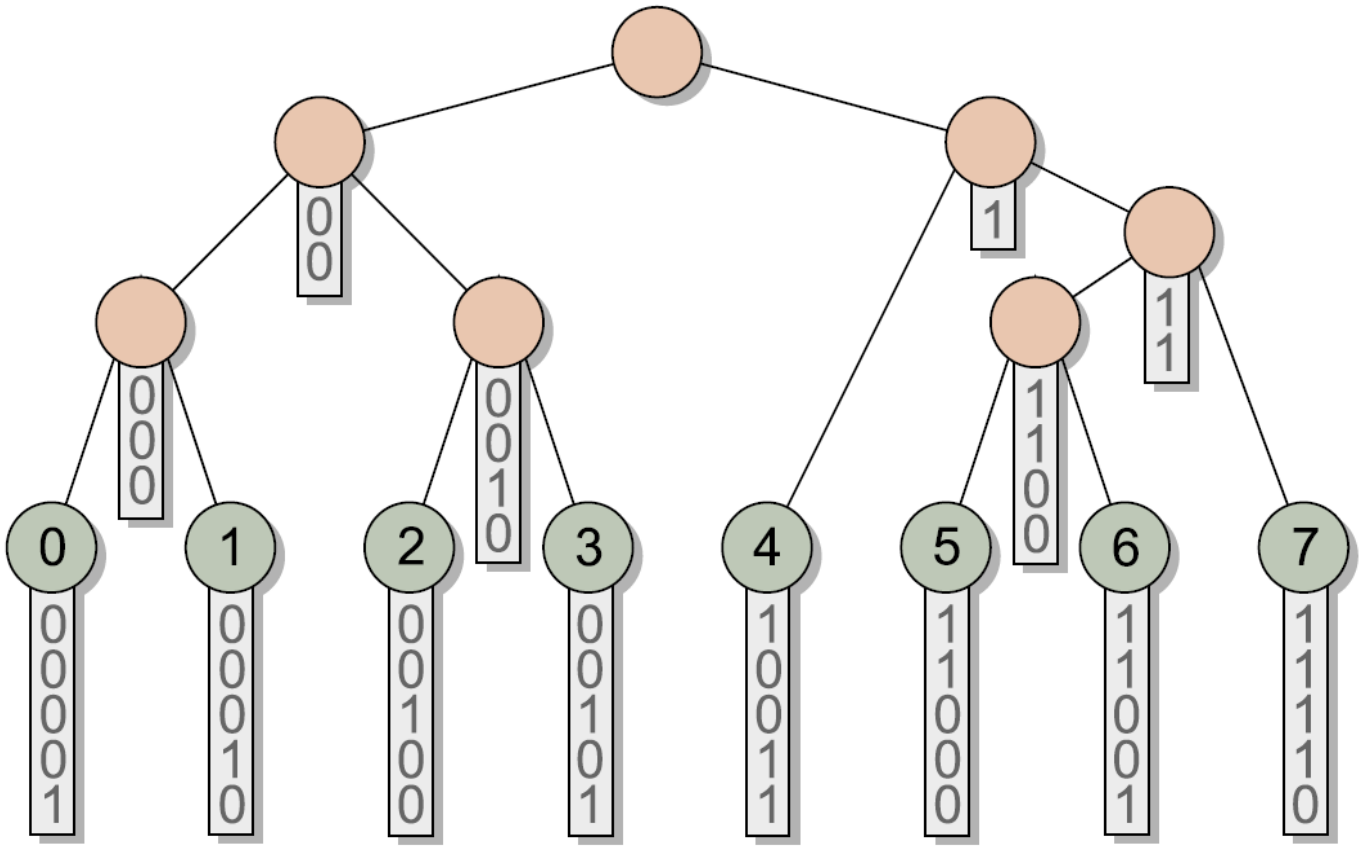
$$\delta(i',j') \geq \delta(i,j)$$



Tero Karras, Maximizing Parallelism in the Construction of BVHs, Octrees, and k-d Trees, High Performance Graphics, 2012

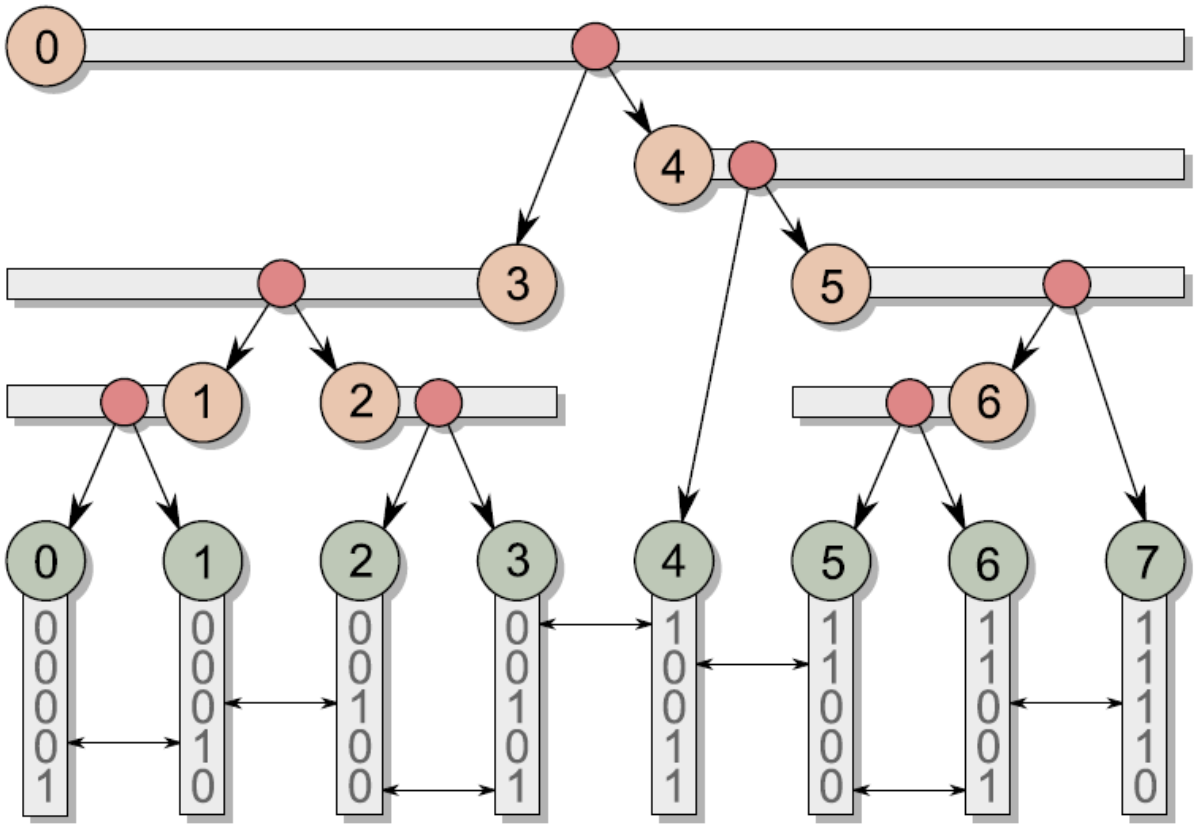
Linear BVH

- 如何并行构建所有内部节点？
  - 自底向上
  - 消除对上层节点依赖



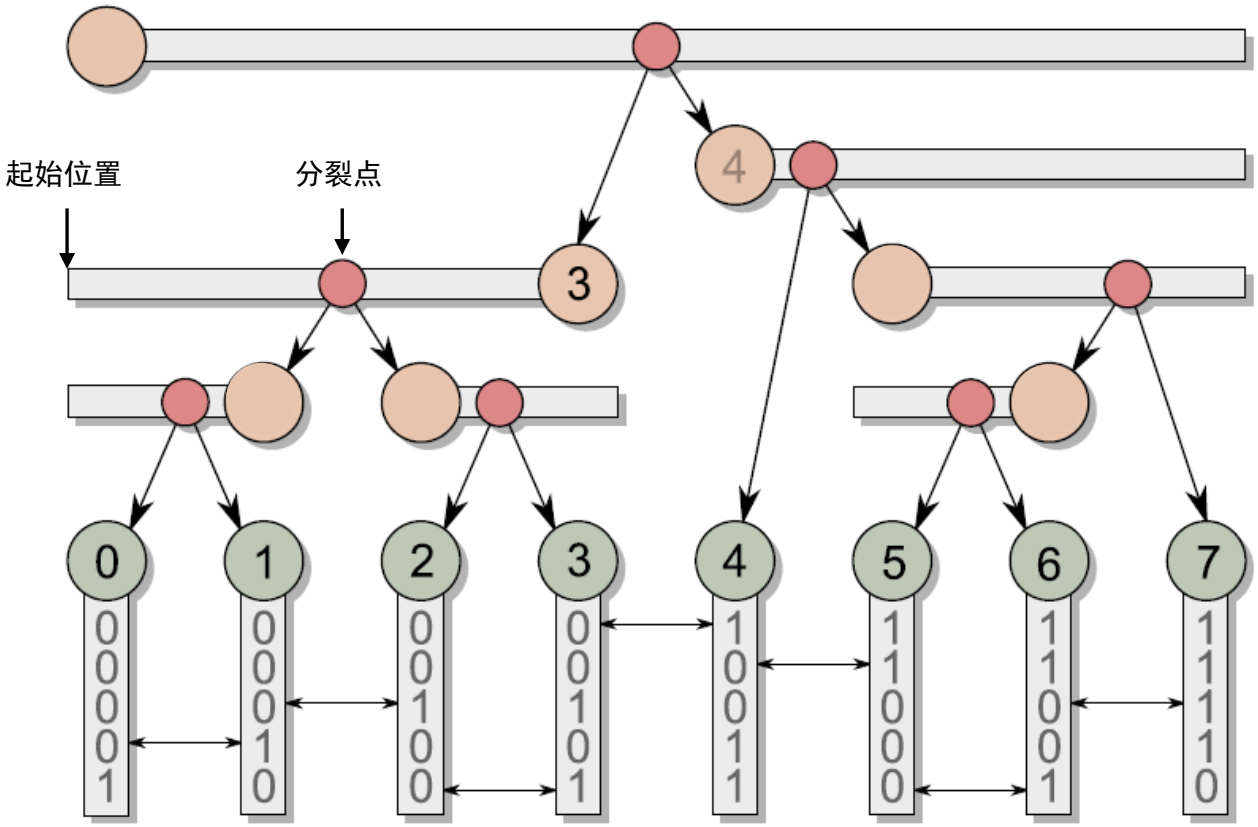
Linear BVH

- Linear BVH特性
  - 假设叶节点数组为，L长度为N，额外分配长度为$N-1$的数组 I 存内部节点
  - 根节点位于I_0
  - 假设某中间节点分割位为γ，则左孩子为I_γ或则L_γ，右孩子为$I_{\gamma+1}$或则$L_{\gamma+1}$
  - 每个内部节点位置与第一个或者最后一个元素位置重合



Linear BVH

- 构建内部节点
  - 计算内部节点索引范围
  - 计算左右子树分裂位置
  - 构建AABB



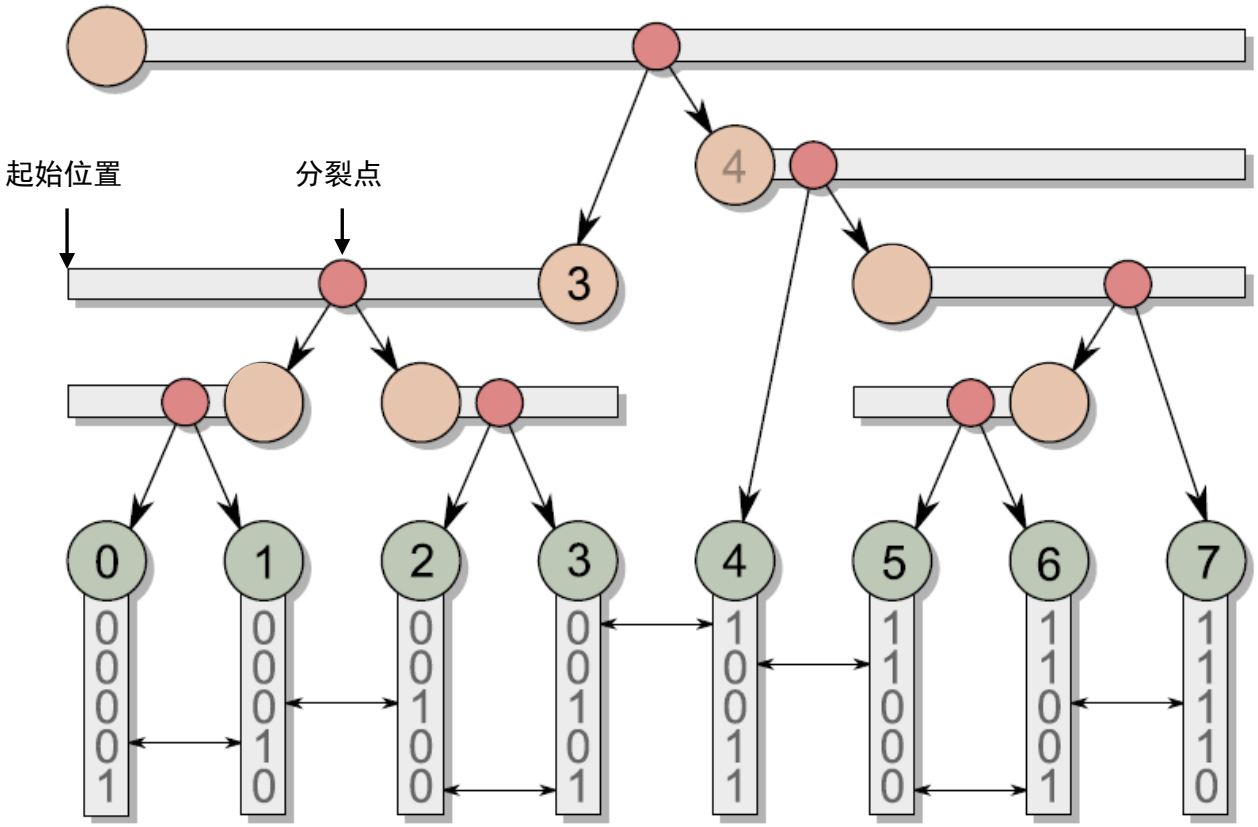
以内部节点3为例

Linear BVH

- 构建过程：确定搜索方向（步骤一）

```
int d = delta(i, i + 1) - delta(i, i - 1) > 0 ? 1 : -1;
```

d > 0 向右搜索
d < 0 向左搜索

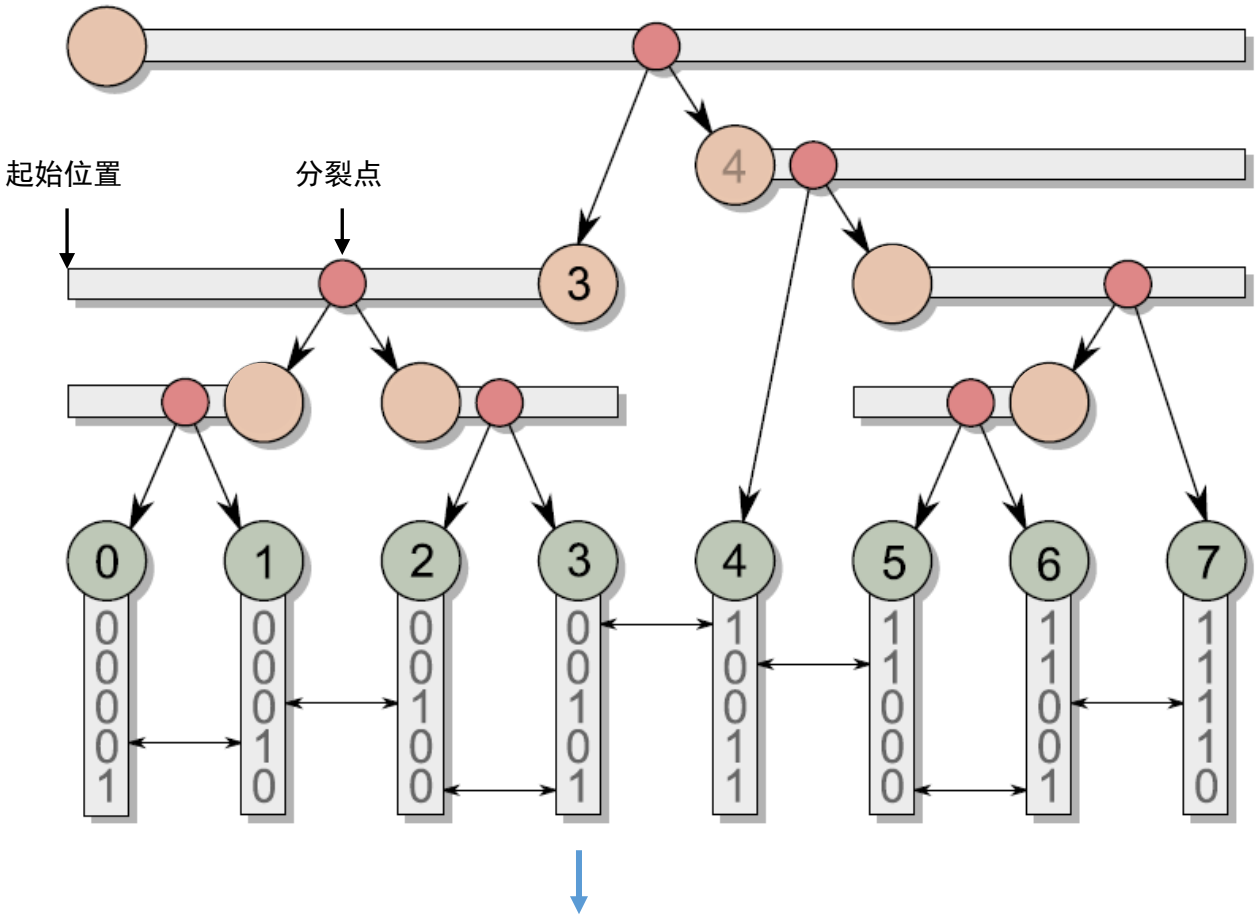


以内部节点3为例

Linear BVH

- 构建过程：查找另一端索引（步骤二） $\delta(i',j') \geq \delta(i,j)$ with $i',j' \in [i,j]$

```
int delta_min = delta(i, i - d);

int len_max = 2;
while (delta(i, i + len_max * d) > delta_min)
{
        len_max *= 2;
}

// Find the other end using binary search
int len = 0;
for (int t = len_max / 2; t > 0; t = t / 2)
{
        if (delta(i, i + (len + t) * d) > delta_min) {
        len = len + t;
        }
}

int j = i + len * d;
```

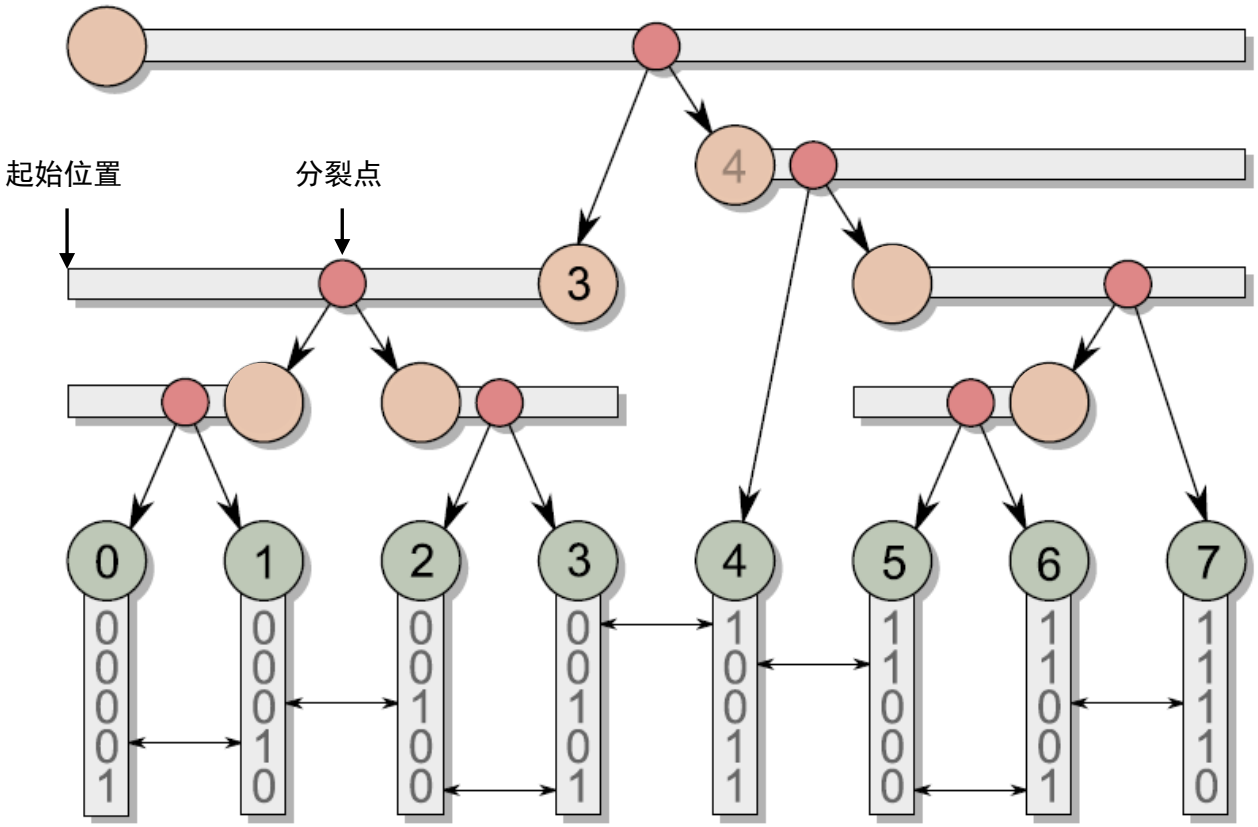


以内部节点3为例

Linear BVH

- 构建过程：计算分裂位置（步骤三 ）

```
int delta_node = delta(i, j);
int s = 0;

for (int t = (len + 1) / 2; t > 0; t = t == 1 ? 0 :
(t + 1) / 2)
{
        if (delta(i, i + (s + t) * d) > delta_node)
        {
                s = s + t;
        }
}
int gamma = i + s * d + minimum(d, (int)0);
```

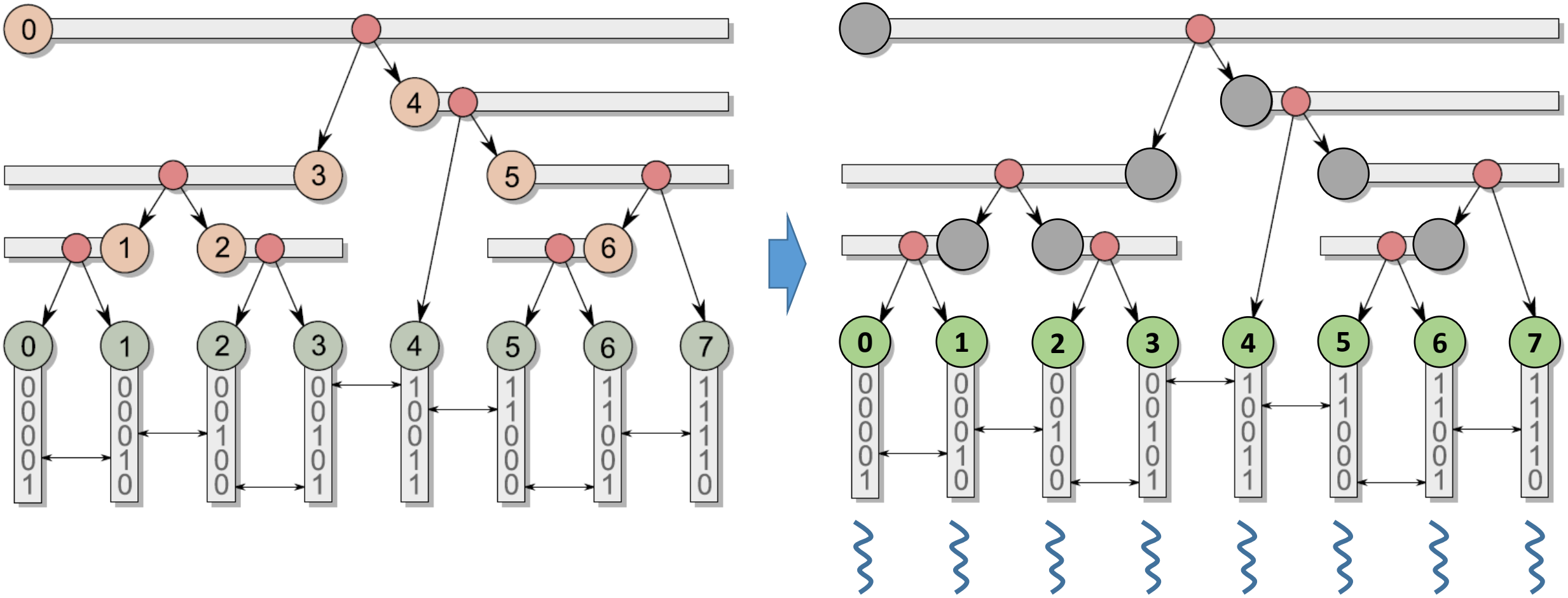


以内部节点3为例

完整实现参见： src/Topology/Cuda/Topology/LinearBVH.cu

Linear BVH

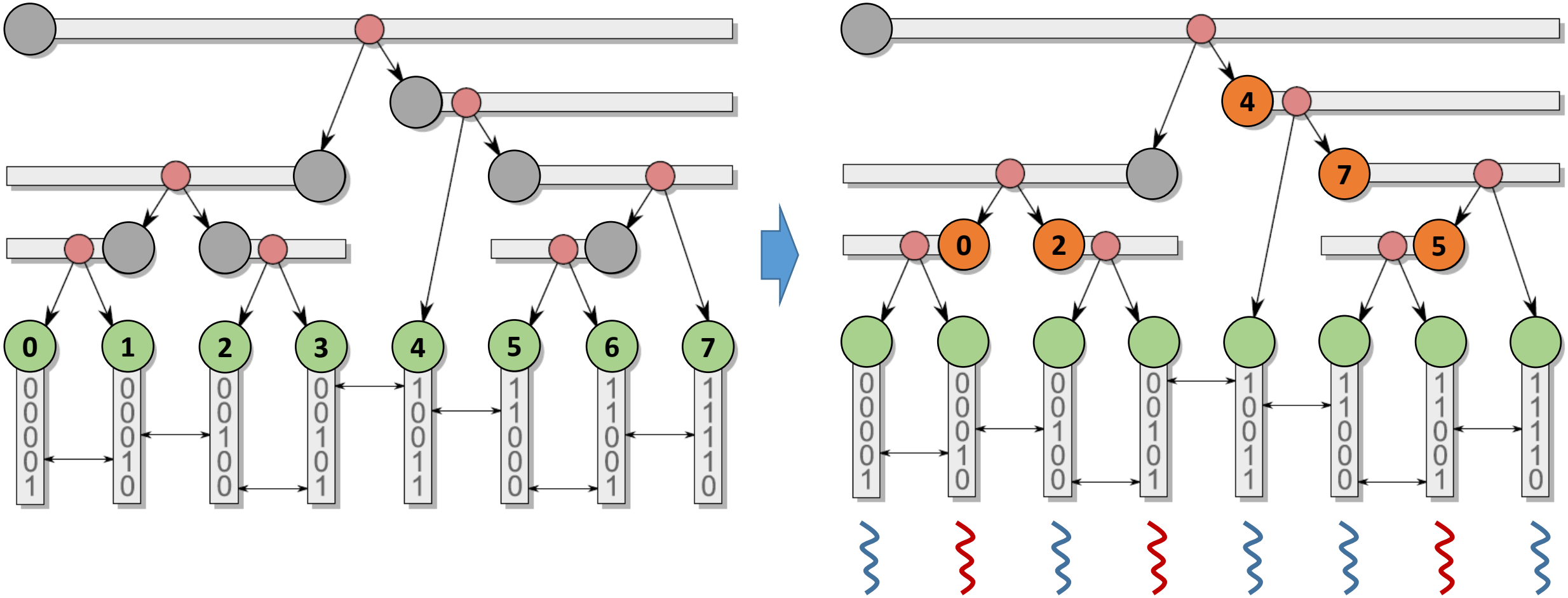
- 构建过程：构建AABB（步骤四）



执行　阻塞　退出　未访问　待定　完成

Linear BVH

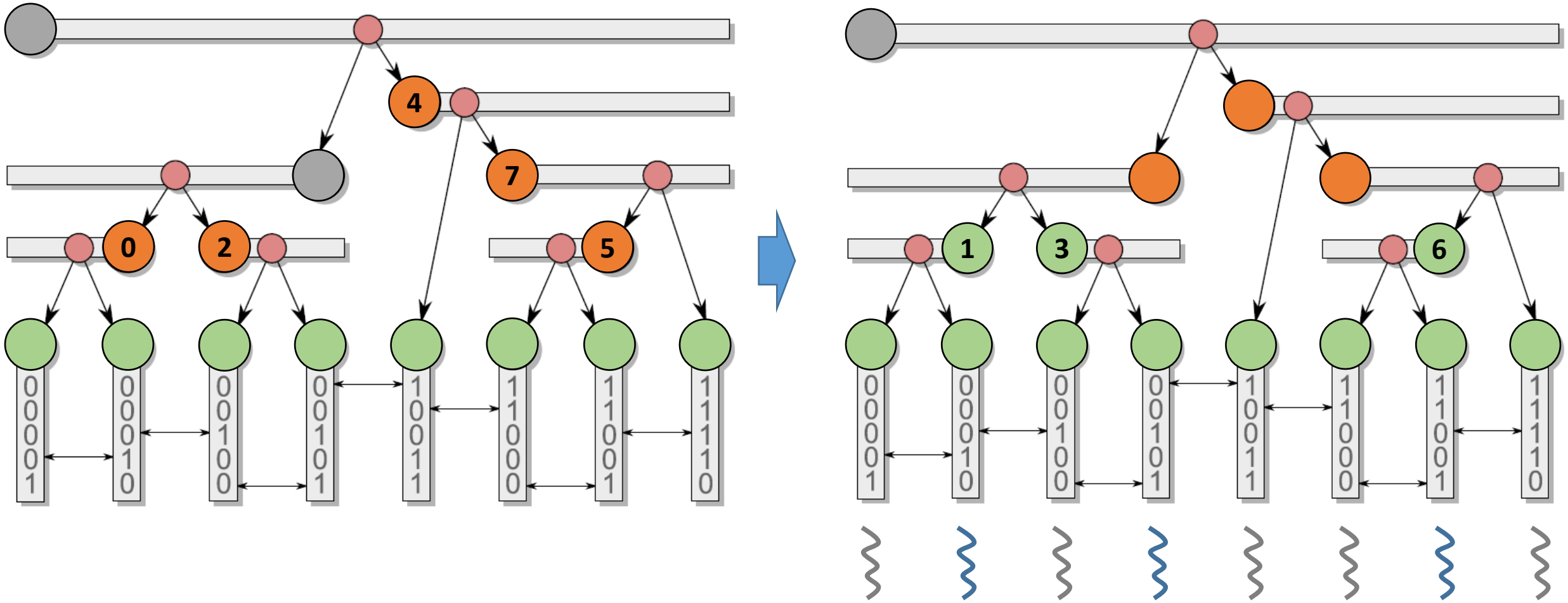
- 构建过程：构建AABB（步骤四）



执行　阻塞　退出　未访问　待定　完成

Linear BVH

- 构建过程：构建AABB（步骤四）



执行　阻塞　退出　未访问　待定　完成

Linear BVH

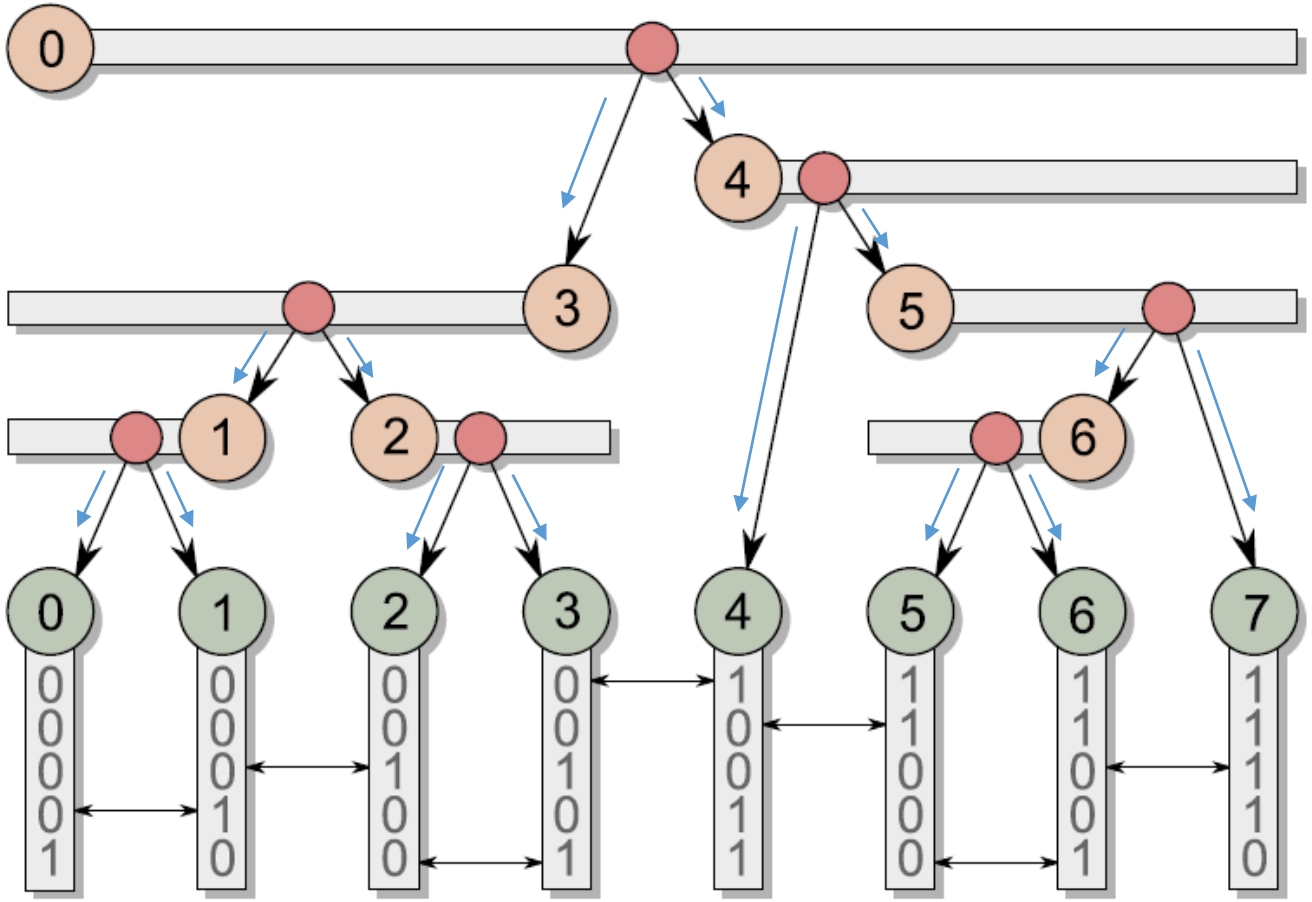
- 构建过程：构建AABB（步骤四）

LBVH_CalculateBoundingBox

```
int idx = bvhNodes[i + N - 1].parent;
while (idx != EMPTY) // means idx == 0
{
    const int old = atomicCAS(flags.begin() + idx, 0, 1);

    if (old == 0) return;

    assert(old == 1);

    const int l_idx = bvhNodes[idx].left;
    const int r_idx = bvhNodes[idx].right;
    const AABB l_aabb = sortedAABBs[l_idx];
    const AABB r_aabb = sortedAABBs[r_idx];
    sortedAABBs[idx] = l_aabb.merge(r_aabb);

    // look the next parent...
    idx = bvhNodes[idx].parent;
}
```

Linear BVH

- 并行遍历（Parallel Traversal）
  - 递归→ Divergence
  - 循环

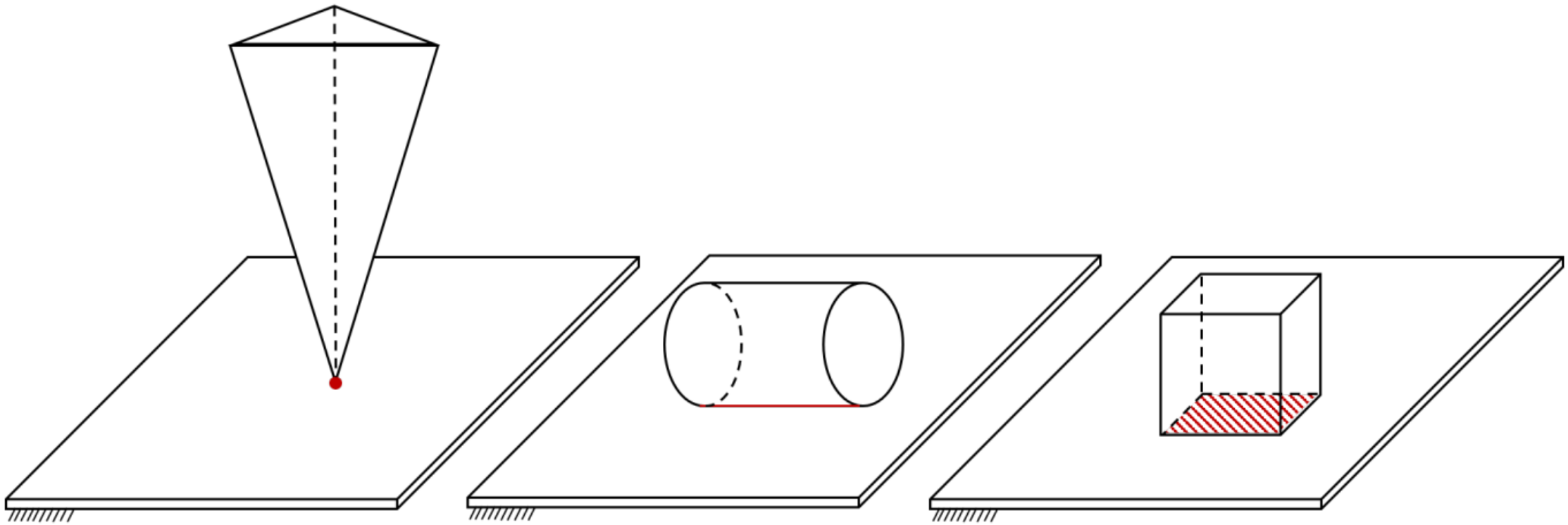


窄阶段碰撞检测

- 接触（Contact）
- 分离轴定理(Separating Axis Theorem)
- 接触流形（Contact Manifold）

接触（Contact)

- 接触类型

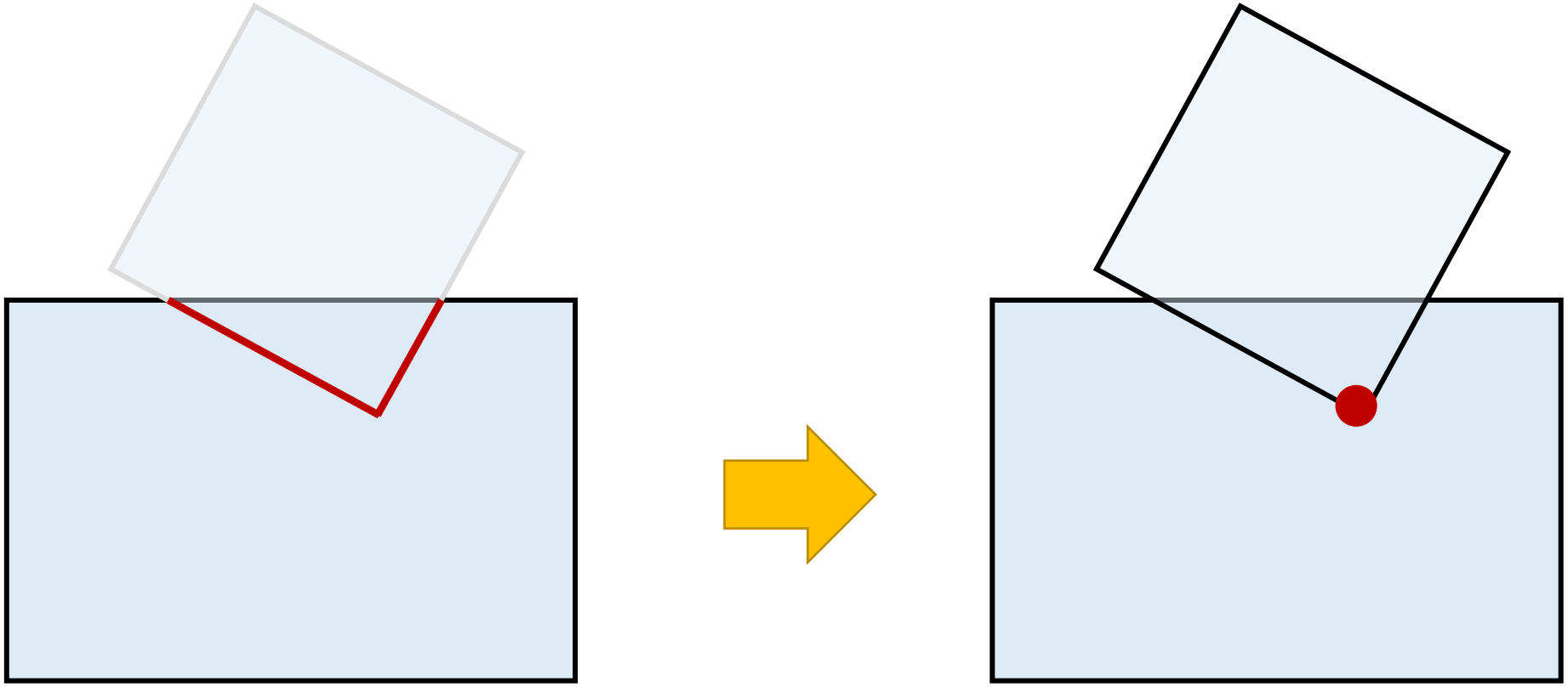


Point Contact　　Line Contact　　Surface Contact

@ arXiv:2010.02291

接触（Contact)

- 接触定义

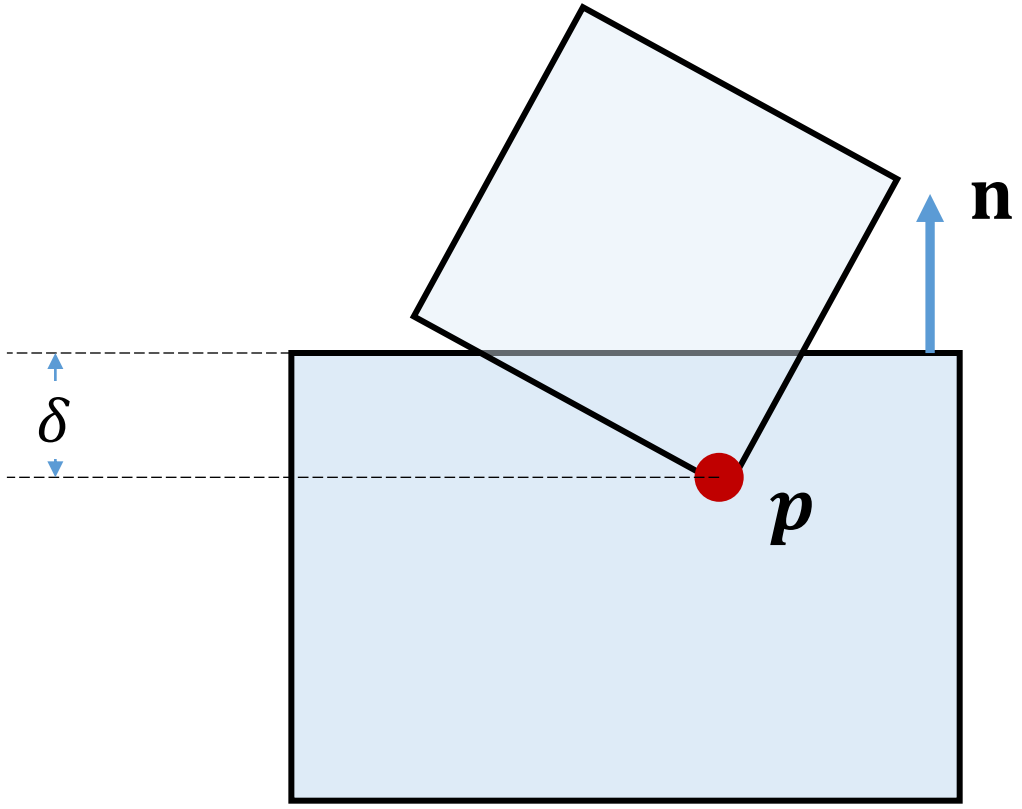


Contact Manifold

Contact Point

接触（Contact）

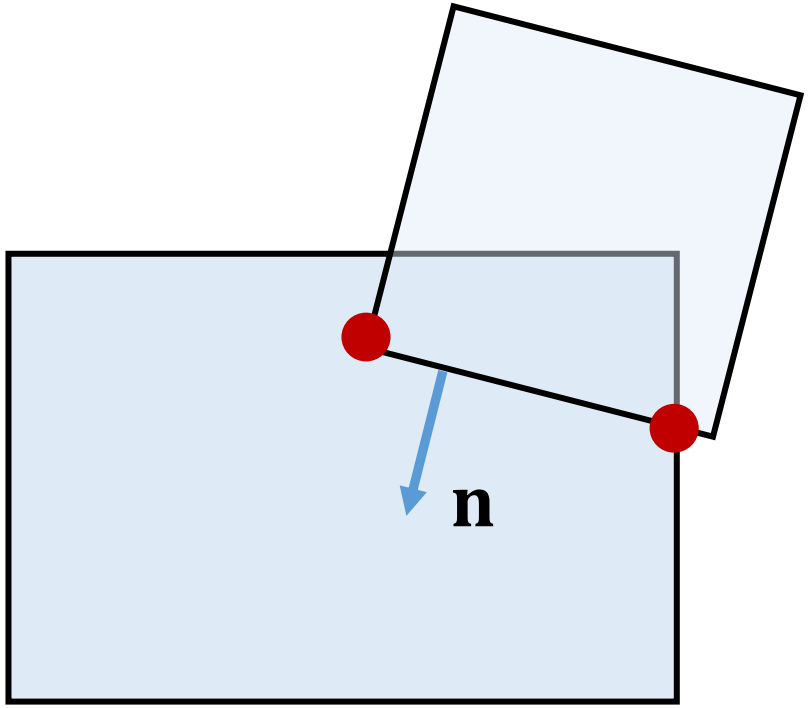
- 触点定义
  - 触点位置
  - 触点ID
  - 法线方向
  - 穿透距离



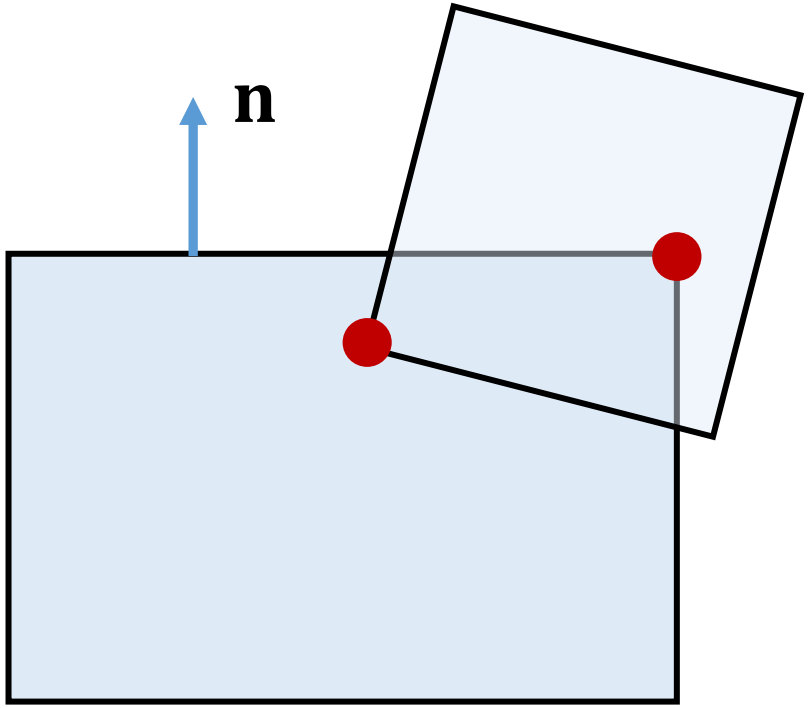
Contact Point

接触（Contact)

- 触点定义歧义性



Contact Points

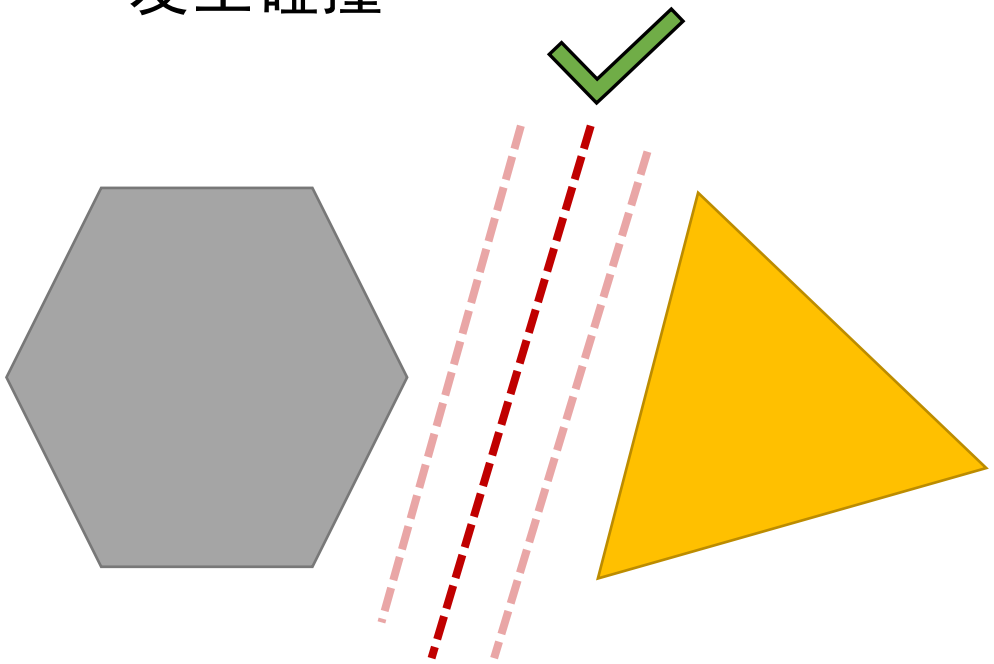


Contact Points

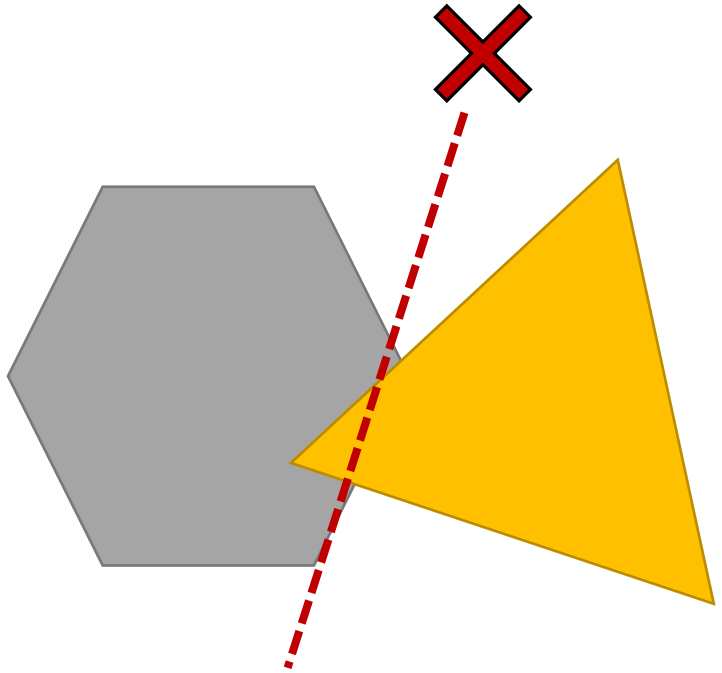
如何保证触点一致性

窄阶段碰撞检测

- 分离轴定理（Separating Axis Theorem）
  - 如果两个物体未发生碰撞，则至少存在一条直线（或平面）将两物体隔离开
  - 反之，如果存在一条直线（或平面）将两物体隔离开，则两物体必定未发生碰撞



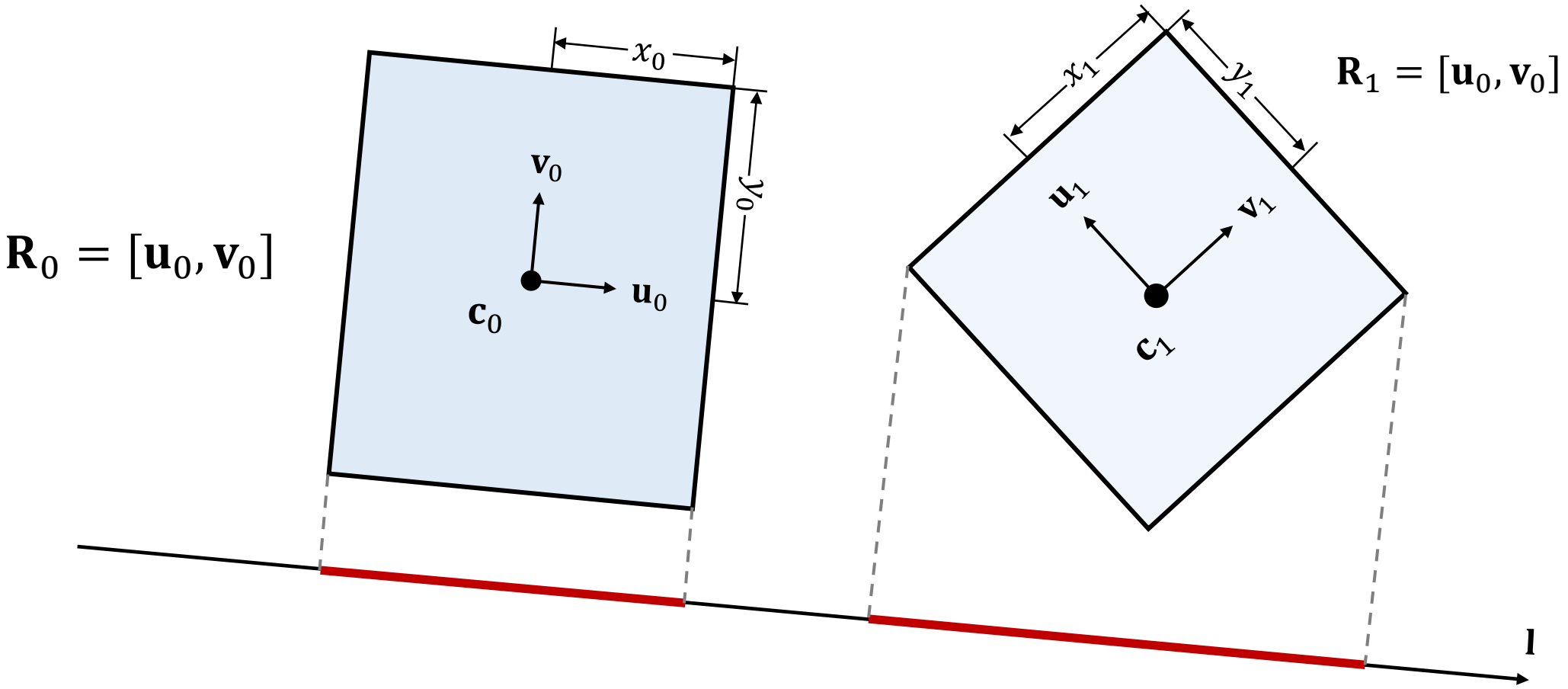
未发生碰撞



发生碰撞

窄阶段碰撞检测

- 分离轴定理（Separating Axis Theorem）



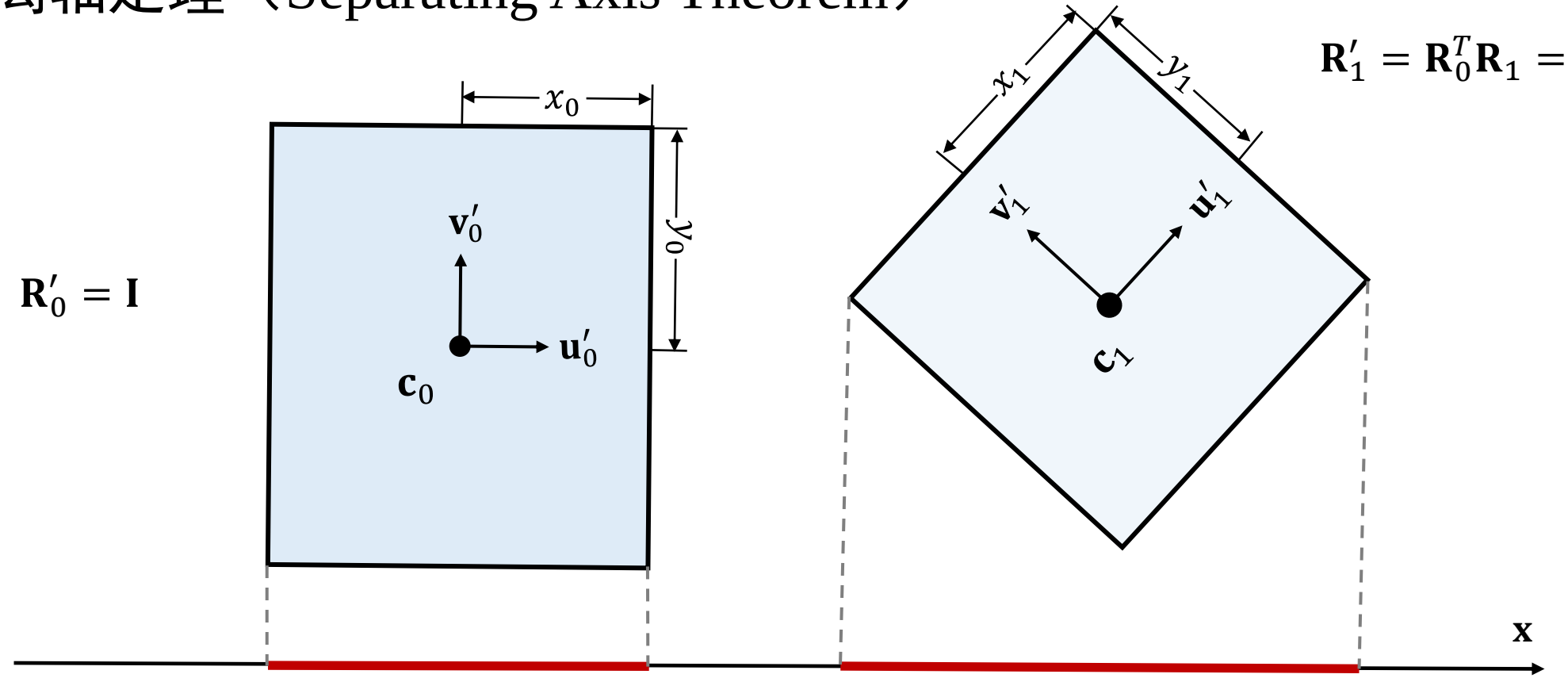
$$|(\mathbf{c}_0 - \mathbf{c}_1) \cdot \mathbf{l}| > x_0|\mathbf{u}_0 \cdot \mathbf{l}| + y_0|\mathbf{v}_0 \cdot \mathbf{l}| + x_1|\mathbf{u}_1 \cdot \mathbf{l}| + y_1|\mathbf{v}_1 \cdot \mathbf{l}|, \qquad \mathbf{l} = \mathbf{u}_0, \mathbf{v}_0, \mathbf{u}_1, \mathbf{v}_1$$

窄阶段碰撞检测

- 分离轴定理（Separating Axis Theorem）

$$\mathbf{R}_1' = \mathbf{R}_0^T\mathbf{R}_1 = [\mathbf{u}_1' \;\; \mathbf{v}_1']$$

$$\mathbf{R}_0' = \mathbf{I}$$



$$|t_x| > x_0 + x_1|\mathbf{u}_{1,x}| + y_1|\mathbf{v}_{1,x}|,$$

$$|t_y| > y_0 + x_1|\mathbf{u}_{1,y}| + y_1|\mathbf{v}_{1,y}|,$$

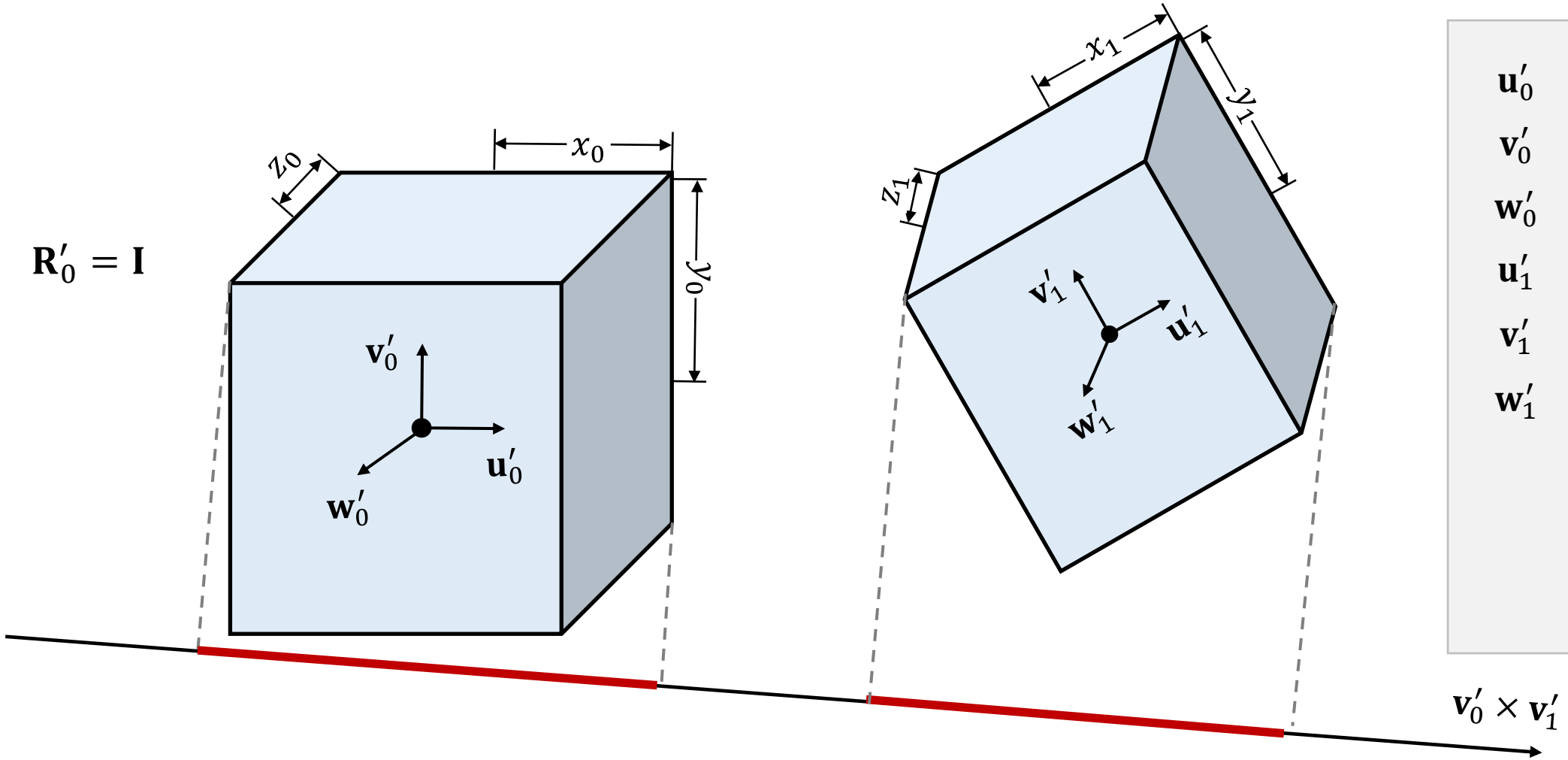
$$\mathbf{t} = \mathbf{R}_0^T(\mathbf{c}_1 - \mathbf{c}_0) = [t_x \;\; t_y]$$

窄阶段碰撞检测

- 分离轴定理（Separating Axis Theorem）

$$\mathbf{R}_1' = \mathbf{R}_0^T \mathbf{R}_1 = [\mathbf{u}_1' \ \ \mathbf{v}_1']$$

$$\mathbf{R}_0' = \mathbf{I}$$



$\mathbf{u}_0'$	$\mathbf{u}_0' \times \mathbf{u}_1'$
$\mathbf{v}_0'$	$\mathbf{u}_0' \times \mathbf{v}_1'$
$\mathbf{w}_0'$	$\mathbf{u}_0' \times \mathbf{w}_1'$
$\mathbf{u}_1'$	$\mathbf{v}_0' \times \mathbf{u}_1'$
$\mathbf{v}_1'$	$\mathbf{v}_0' \times \mathbf{v}_1'$
$\mathbf{w}_1'$	$\mathbf{v}_0' \times \mathbf{w}_1'$
	$\mathbf{w}_0' \times \mathbf{u}_1'$
	$\mathbf{w}_0' \times \mathbf{v}_1'$
	$\mathbf{w}_0' \times \mathbf{w}_1'$

窄阶段碰撞检测

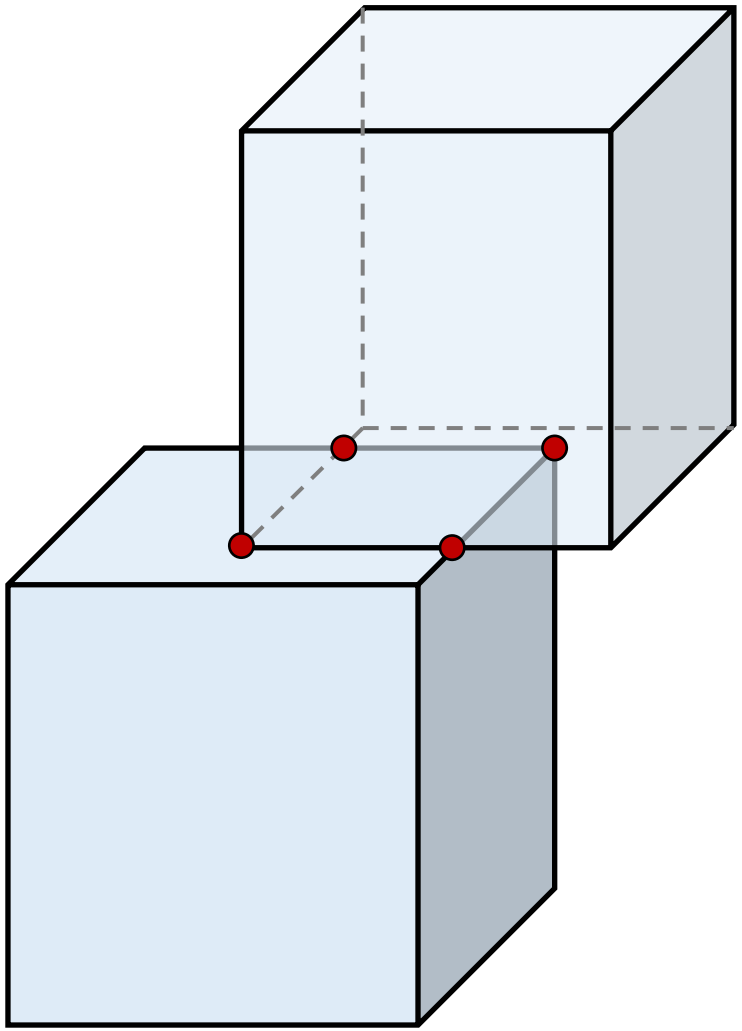
- 接触流形（Contact Manifold）

Line Contact

Surface Contact

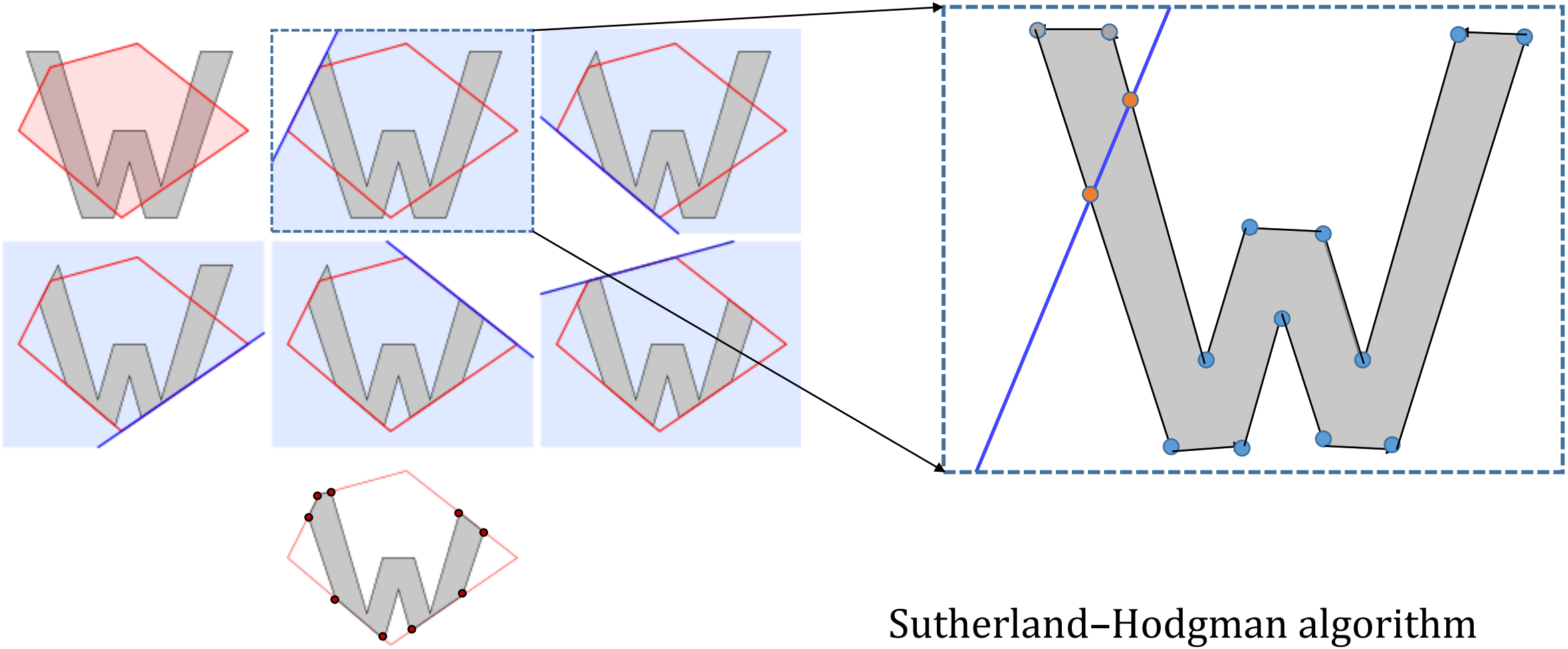
↓

Contact Points



窄阶段碰撞检测

- 接触流形（Contact Manifold）



Sutherland–Hodgman algorithm

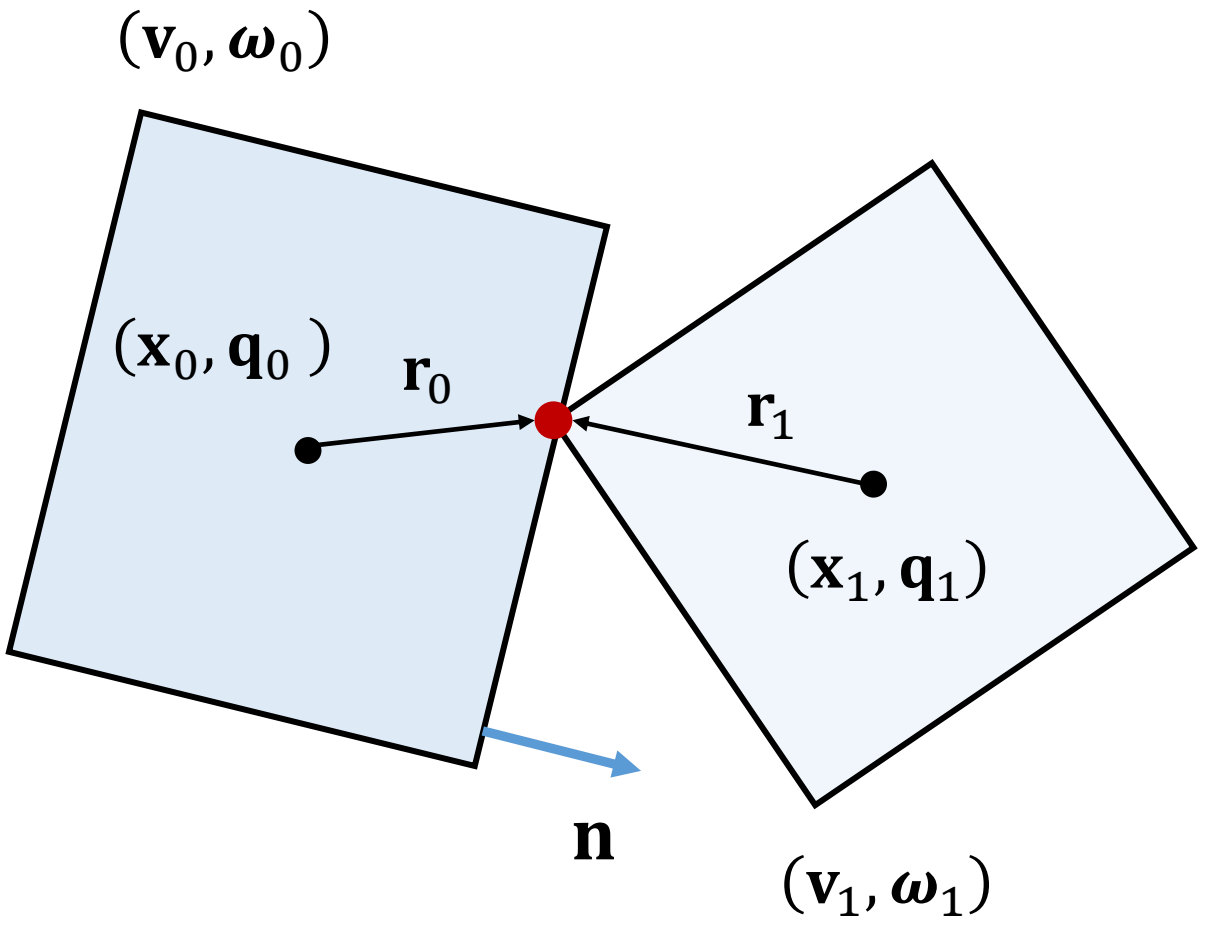
刚体动力学求解

- 速度约束

$$\dot{C} = (\mathbf{v}_{r_1} - \mathbf{v}_{r_0}) \cdot \mathbf{n} < 0 \begin{cases} > 0, & \text{分离} \\ = 0, & \text{接触} \\ < 0, & \text{穿透} \end{cases}$$

$$\mathbf{v}_{r_0} = \mathbf{v}_0 + \boldsymbol{\omega}_0 \times \mathbf{r}_0$$

$$\mathbf{v}_{r_1} = \mathbf{v}_1 + \boldsymbol{\omega}_1 \times \mathbf{r}_1$$



刚体动力学求解

- 速度约束

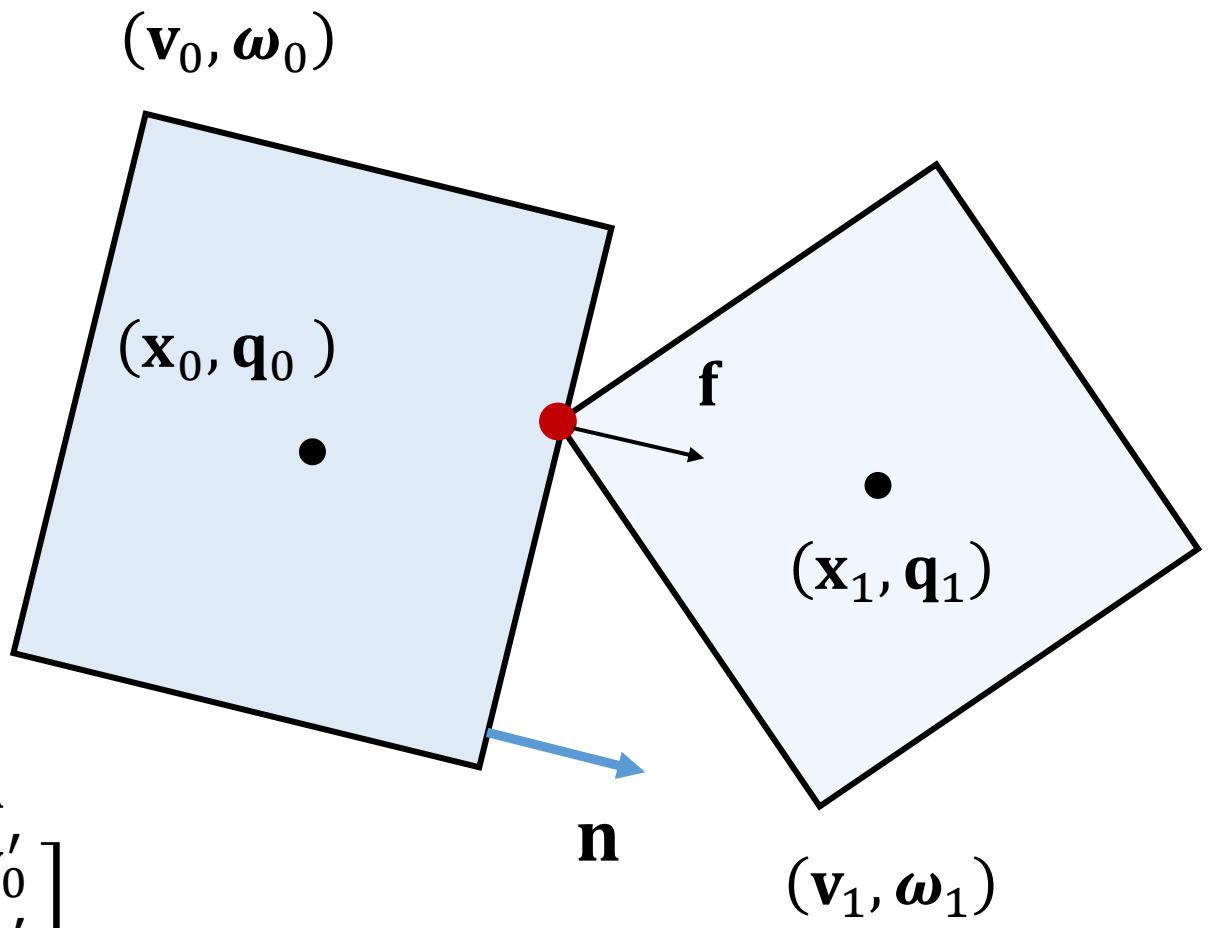
$$C = \left(\mathbf{v}'_{r_1} - \mathbf{v}'_{r_0}\right) \cdot \mathbf{n} = 0$$

$$\mathbf{v}'_{r_0} = \mathbf{v}'_0 + \boldsymbol{\omega}'_0 \times \boldsymbol{r}'_0$$

$$\mathbf{v}'_{r_1} = \mathbf{v}'_1 + \boldsymbol{\omega}'_1 \times \boldsymbol{r}'_1$$

$$C = (\mathbf{v}'_1 + \boldsymbol{\omega}'_1 \times \boldsymbol{r}'_1) \cdot \mathbf{n} - (\mathbf{v}'_0 + \boldsymbol{\omega}'_0 \times \boldsymbol{r}'_0) \cdot \mathbf{n}$$

$$= \underbrace{\begin{bmatrix} -\mathbf{n}^T & -(\mathbf{r}_0 \times \mathbf{n})^T & \mathbf{n}^T & (\mathbf{r}_1 \times \mathbf{n})^T \end{bmatrix}}_{\mathbf{J}:\,Jacobian} \underbrace{\begin{bmatrix} \mathbf{v}'_0 \\ \boldsymbol{\omega}'_0 \\ \mathbf{v}'_1 \\ \boldsymbol{\omega}'_1 \end{bmatrix}}_{\mathbf{V}'} = \mathbf{J}\mathbf{V}' = 0$$



刚体动力学求解

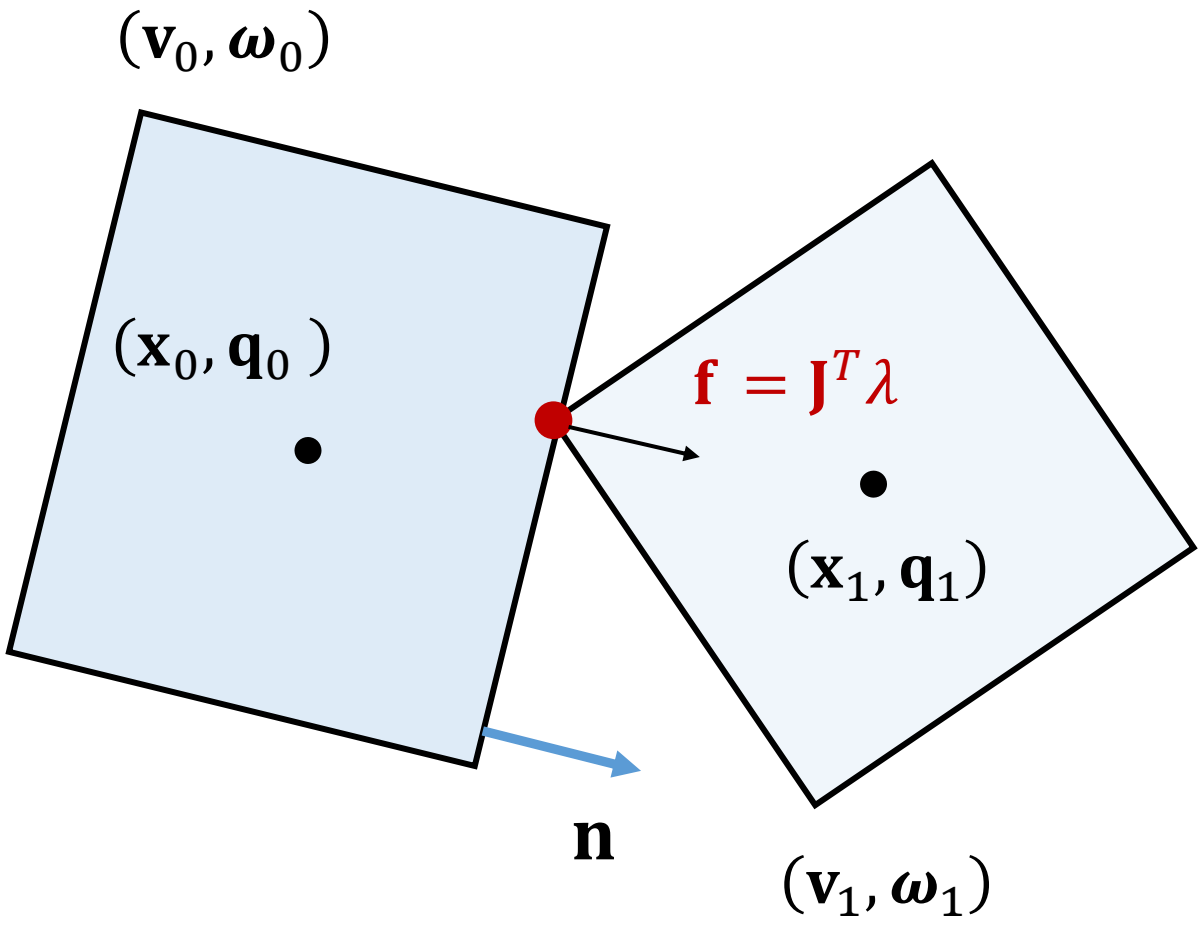
- 接触力
  - 不对系统做功 $\mathbf{f} \cdot \mathbf{V}' = 0$

$$\mathbf{f} \cdot \mathbf{V}' = (\mathbf{J}^T \lambda) \cdot \mathbf{V}' \longleftarrow \mathbf{f} = \mathbf{J}^T \lambda$$

$$= \lambda^T \cdot \mathbf{J}\mathbf{V}' \longleftarrow \mathbf{J}\mathbf{V}' = 0$$

$$= \lambda^T \cdot \mathbf{0}$$

$$= \mathbf{0}$$



刚体动力学求解

- 线性方程组推导

$$\mathbf{J}\mathbf{V}' = 0$$

$$\mathbf{M}\dot{\mathbf{V}}' = \mathbf{J}^T\lambda + \boldsymbol{f}$$

$$\dot{\mathbf{V}}' \approx \frac{\mathbf{V}' - \mathbf{V}}{\Delta t}$$

$$\mathbf{M}\left(\frac{\mathbf{V}' - \mathbf{V}}{\Delta t}\right) = \mathbf{J}^T\lambda + \boldsymbol{f}$$

$$\mathbf{V}' = \mathbf{V} + \Delta t\mathbf{M}^{-1}(\mathbf{J}^T\lambda + \boldsymbol{f})$$

$$\mathbf{J}\mathbf{V} + \Delta t\mathbf{J}\mathbf{M}^{-1}(\mathbf{J}^T\lambda + \boldsymbol{f}) = 0$$

$$\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\lambda = -\frac{1}{\Delta t}\mathbf{J}\left(\mathbf{V} + \mathbf{M}^{-1}\boldsymbol{f}\right) \Leftrightarrow \mathbf{A}\mathbf{x} = \mathbf{b}$$

$$\mathbf{x}' = \mathbf{x} + \Delta t\mathbf{v}'$$

$$\boldsymbol{q}' = \boldsymbol{q} + \frac{\Delta t}{2}(\boldsymbol{\omega}', 0) \times \boldsymbol{q}$$

刚体动力学求解

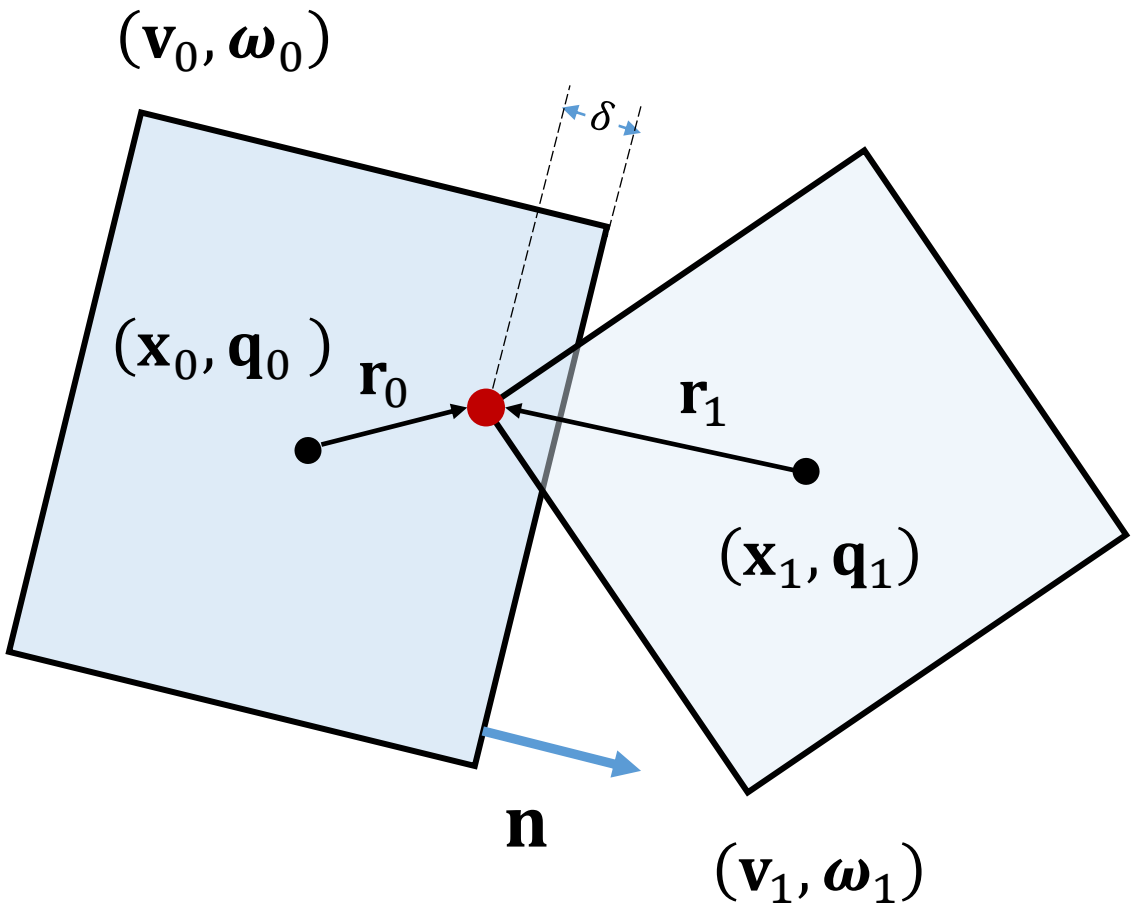
- 已经发生穿透怎么办？

Velocity-based method

$$\mathbf{J}\mathbf{V}' + \boldsymbol{b} = \mathbf{0}\,, \boldsymbol{b} = \frac{\beta}{\Delta t}\boldsymbol{C}, \quad \beta \in [0,1]$$

Position-based method

- ✓ [**Müller et al. 2020**]: Detailed Rigid Body Simulation with Extended Position Based Dynamics
- ✓ [**Lan et al. 2022**]: Affine Body Dynamics: Fast, Stable and Intersection-free Simulation of Stiff Materials

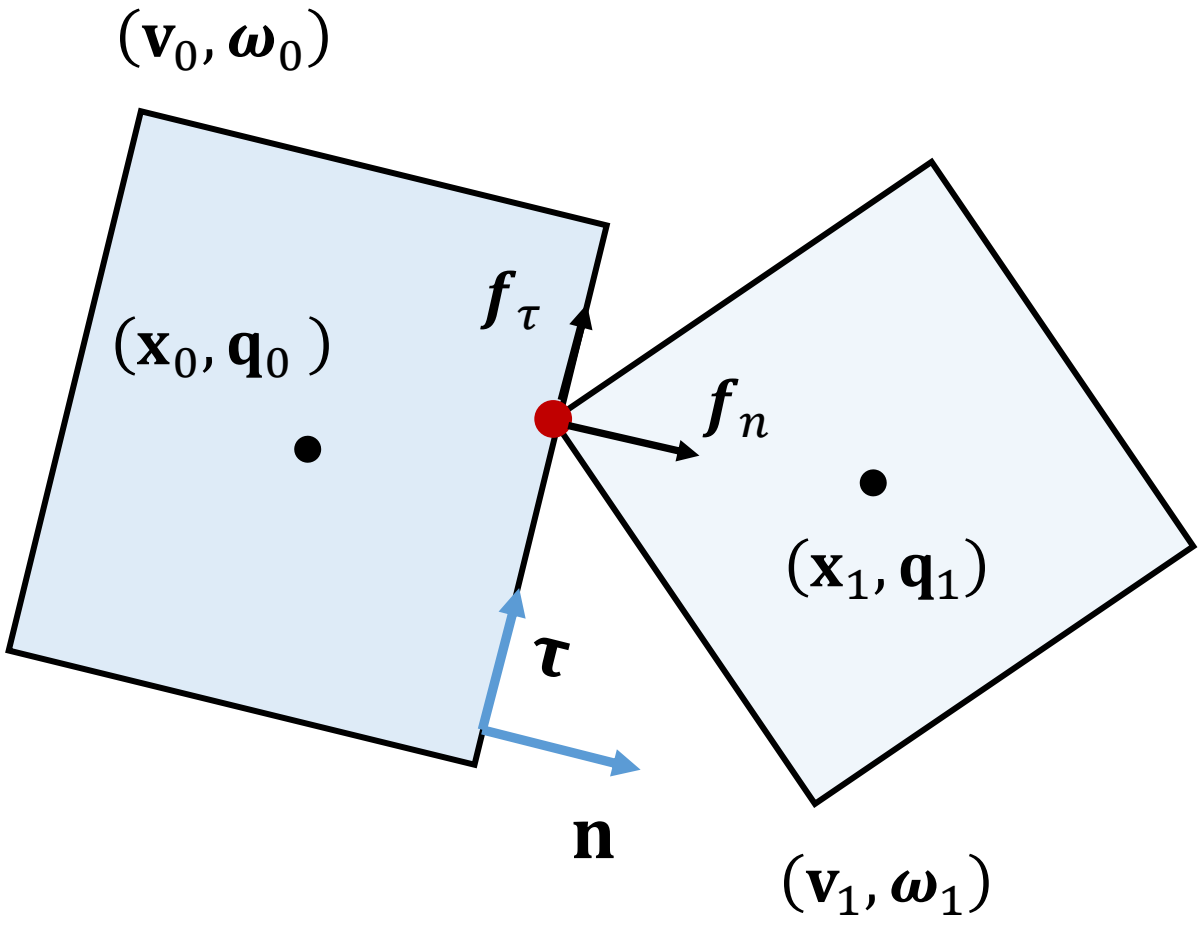


刚体动力学求解

- Friction Constraint：

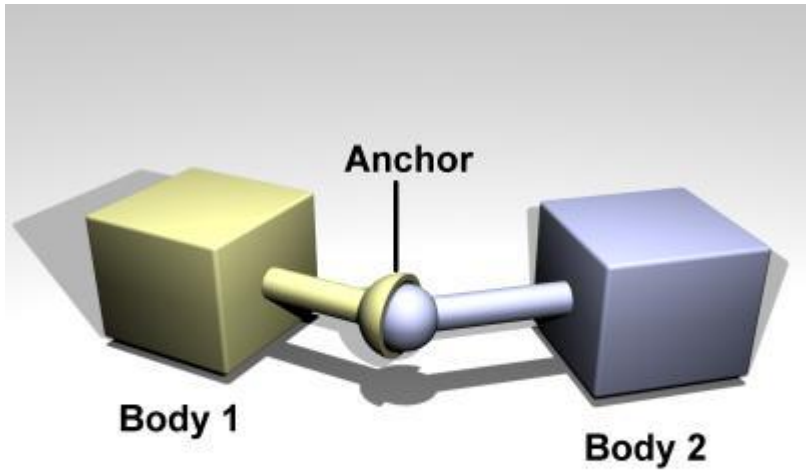
$$\boldsymbol{C}_{fric} = \left(\mathbf{v}_{r_1} - \mathbf{v}_{r_0}\right) \cdot \boldsymbol{\tau} = \mathbf{0}$$

with $\|\boldsymbol{f}_\tau\| \leq \mu \|\boldsymbol{f}_n\|$

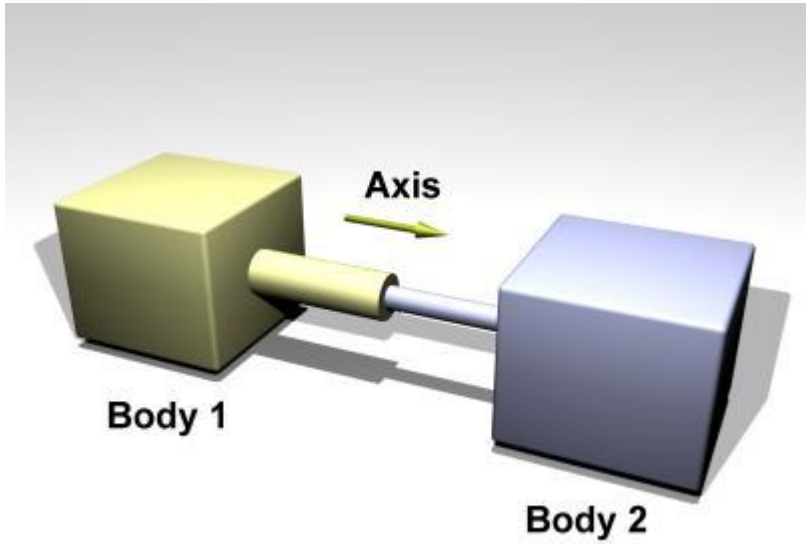


刚体动力学求解

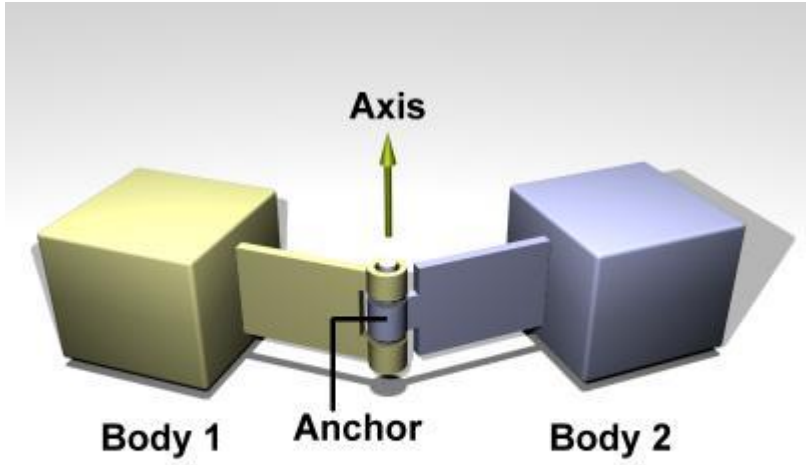
- 常见约束类型



Ball-And-Socket Joint
(3DOF)



Piston Joint
(2 DOF)

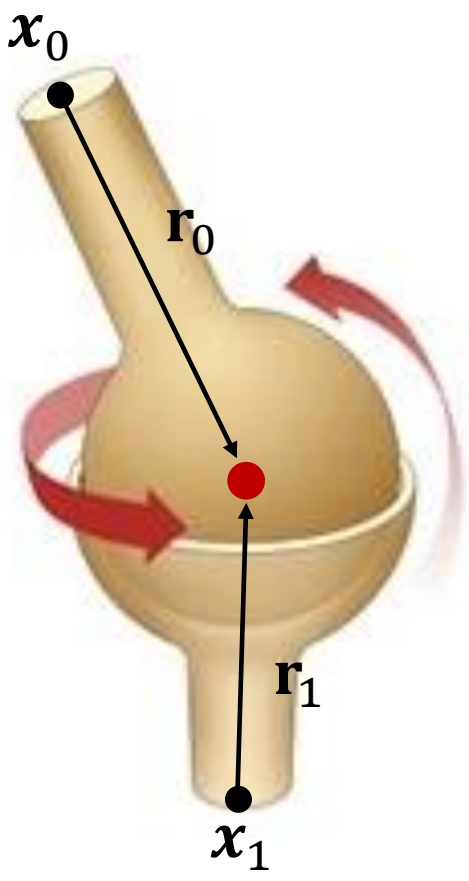


Hinge Joint
(1 DOF)

刚体动力学求解

- Ball-And-Socket Joint

$$\boldsymbol{C}_{trans} = \boldsymbol{x}_1 + \boldsymbol{r}_1 - \boldsymbol{x}_0 + \boldsymbol{r}_0 = \boldsymbol{0}$$

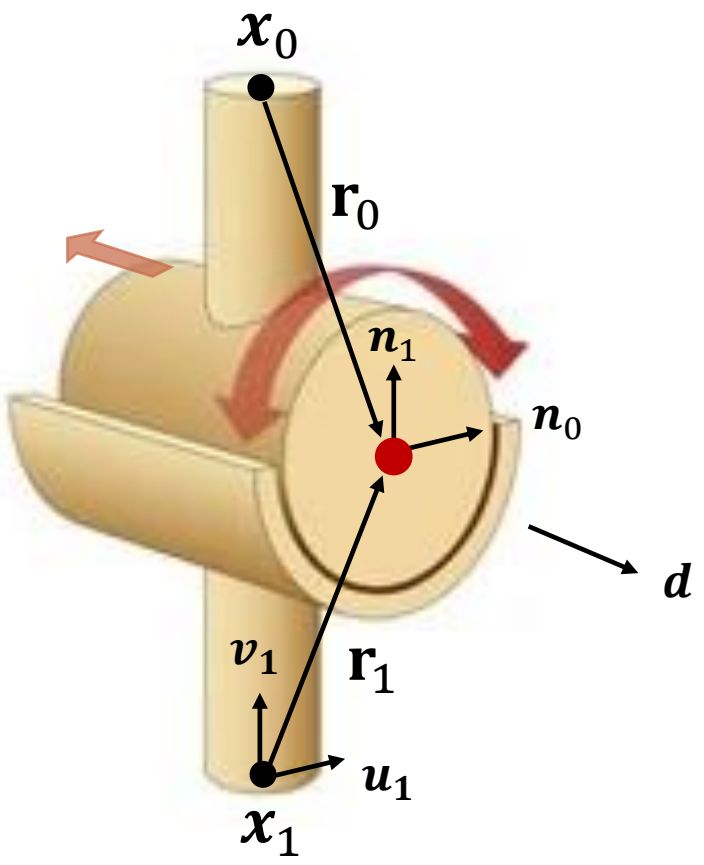


刚体动力学求解

- Piston Joint:

$$\boldsymbol{C}_{trans} = \begin{pmatrix} (\boldsymbol{x}_1 + \boldsymbol{r}_1 - \boldsymbol{x}_0 + \boldsymbol{r}_0) \cdot \boldsymbol{n}_0 \\ (\boldsymbol{x}_1 + \boldsymbol{r}_1 - \boldsymbol{x}_0 + \boldsymbol{r}_0) \cdot \boldsymbol{n}_1 \end{pmatrix} = \boldsymbol{0}$$

$$\boldsymbol{C}_{rot} = \begin{pmatrix} \boldsymbol{d} \cdot \boldsymbol{u}_1 \\ \boldsymbol{d} \cdot \boldsymbol{v}_1 \end{pmatrix} = \boldsymbol{0}$$

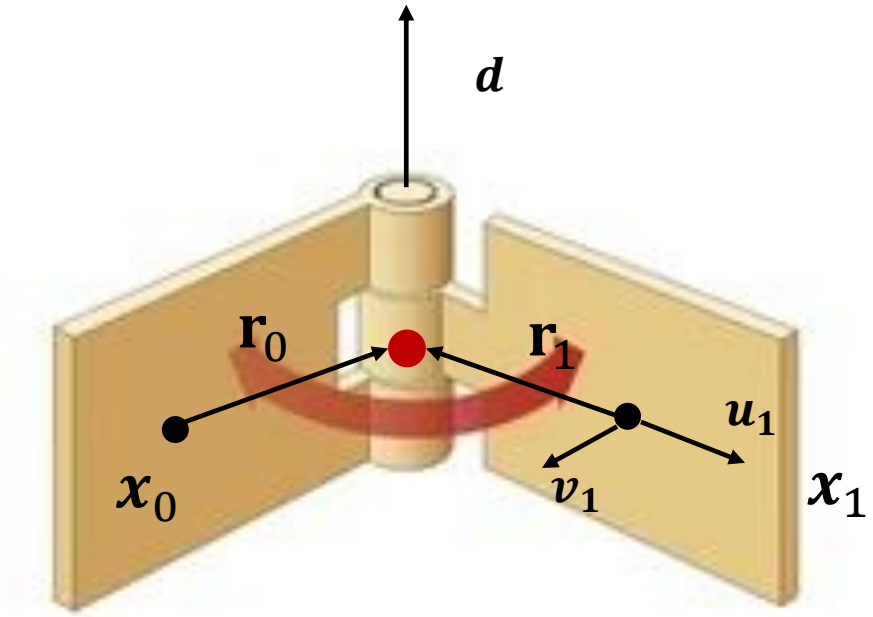


刚体动力学求解

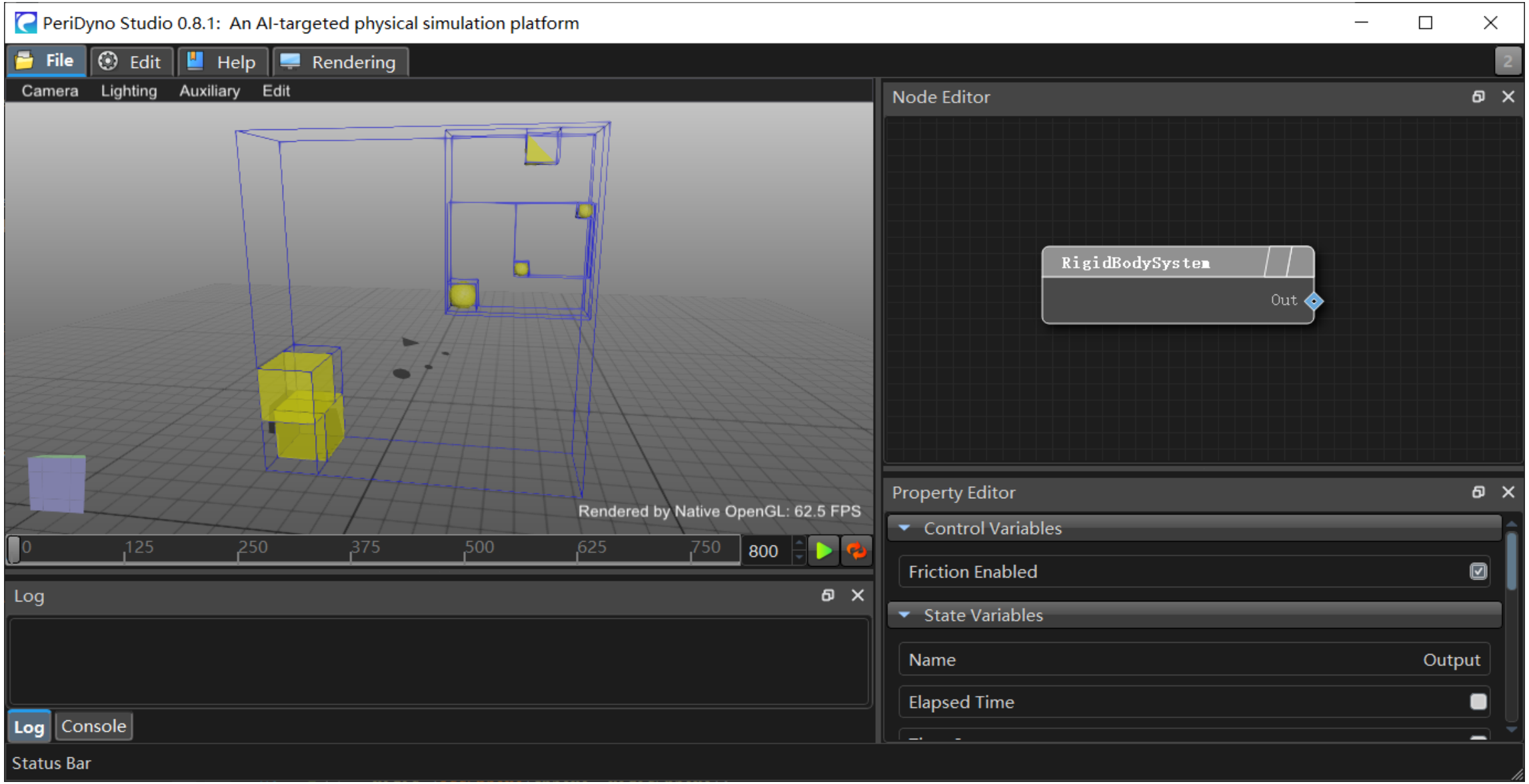
- Hinge Joint:

$$\boldsymbol{C}_{trans} = \boldsymbol{x}_1 + \boldsymbol{r}_1 - \boldsymbol{x}_0 + \boldsymbol{r}_0 = \boldsymbol{0}$$

$$\boldsymbol{C}_{rot} = \begin{pmatrix} \boldsymbol{d} \cdot \boldsymbol{u}_1 \\ \boldsymbol{d} \cdot \boldsymbol{v}_1 \end{pmatrix} = \boldsymbol{0}$$

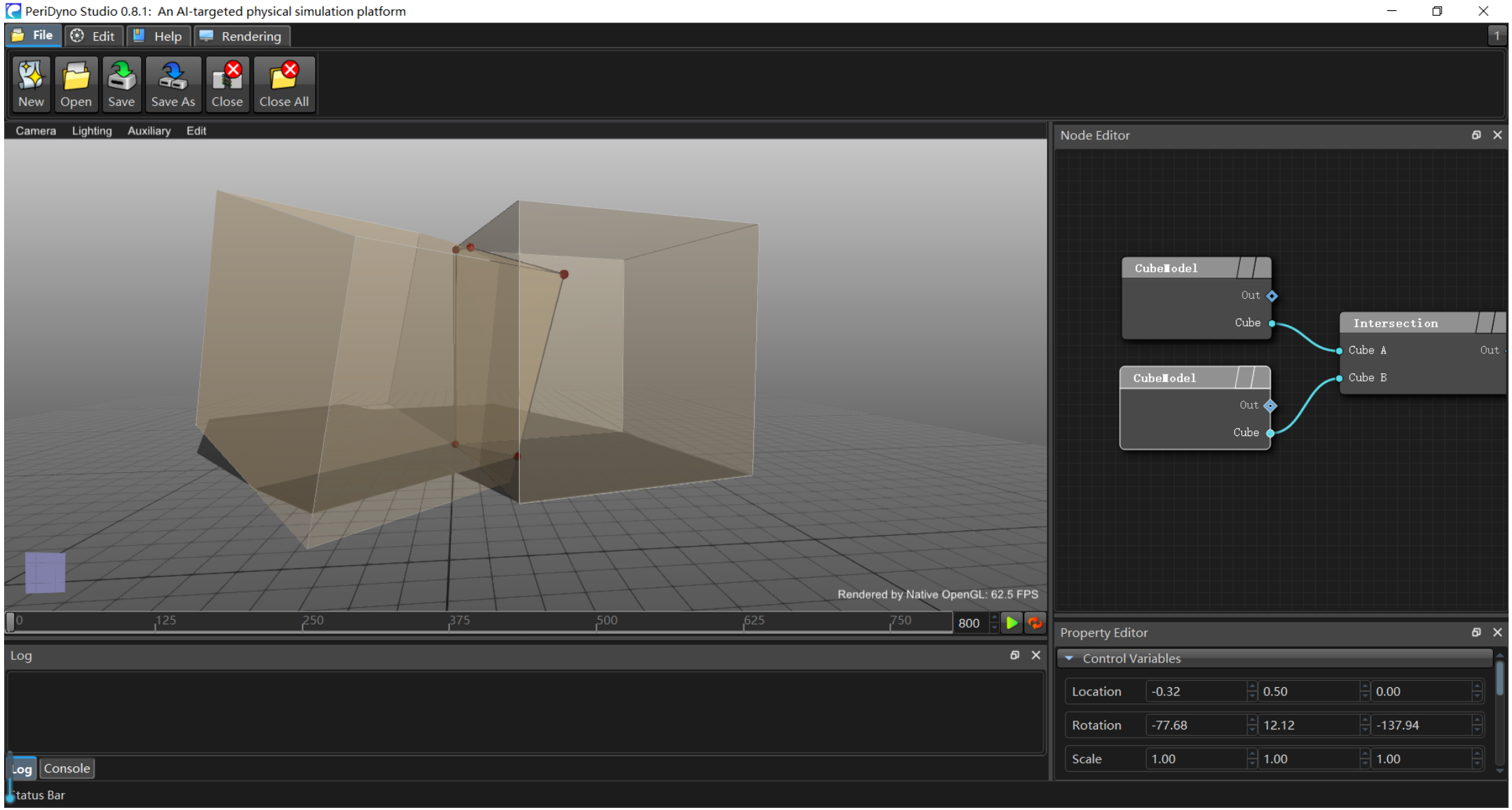


演示场景一： Linear BVH



example/Cuda/RigidBody/Qt_LinearBVH

场景演示二：SAT



example/Cuda/RigidBody/ Qt_SAT

参考资料

- https://developer.nvidia.com/blog/author/tkarras/
- https://luebke.us/publications/eg09.pdf
- http://ode.org/wikiold/htmlfile15.html
- https://en.wikipedia.org/wiki/Sutherland%E2%80%93Hodgman_algorithm
- Daniel Chappuis, Constraints Derivation for Rigid Body Simulation in 3D, 2013

Question