

# Физика вблизи скорости света: визуализация эффектов СТО

# Реализованные эффекты

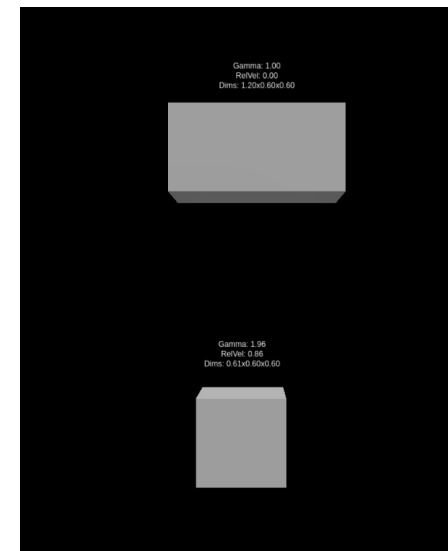
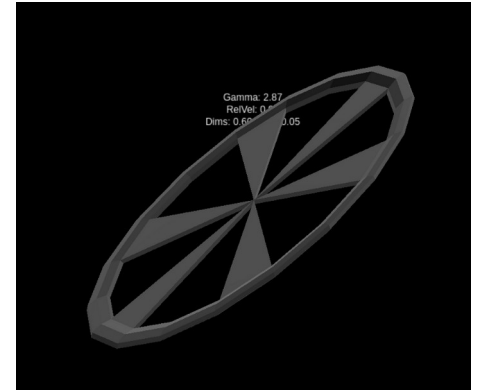
Используя преобразования Лоренца как фундамент для движка, удалось наглядно продемонстрировать ряд релятивистских эффектов:

- Лоренцево замедление времени
- Лоренцево сокращение длины

Эти эффекты можно увидеть на примере следующих экспериментов:

- 1) Линейные рамеры параллелепипедов
- 2) Релятивистские гонки
- 3) Объекты сложной геометрии
- 4) Город на скоростях света

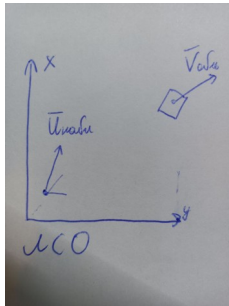
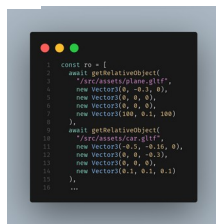
<http://localhost:5173/index.html>



# Принцип работы

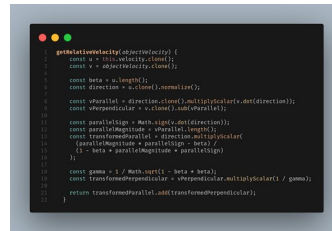
## Создание объектов

- 1) Собственные размеры
- 2) Скорость в лсо



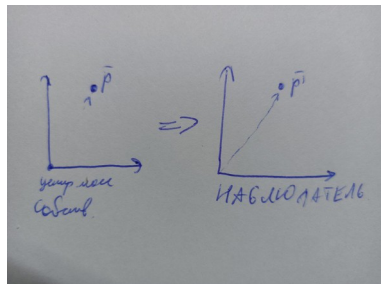
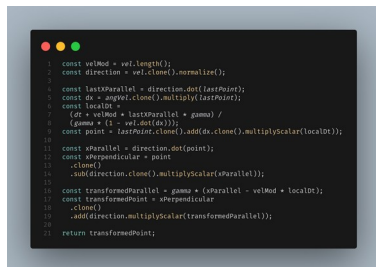
## Переход в систему отсчета наблюдателя

- 1) Расчет относительной скорости для каждого объекта
- 2) Расчет центра масс каждого объекта



## Пересчет каждой точки из собственной системы отсчета в систему отсчета наблюдателя

- 1) Преобразования Лоренца
- 2) Перенос центра координат в другую точку (для простоты)

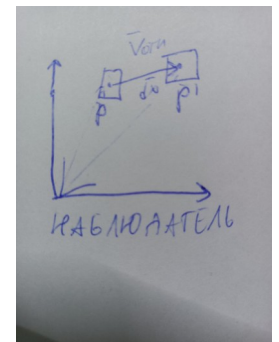


## Dt наблюдателя (каждый тик)

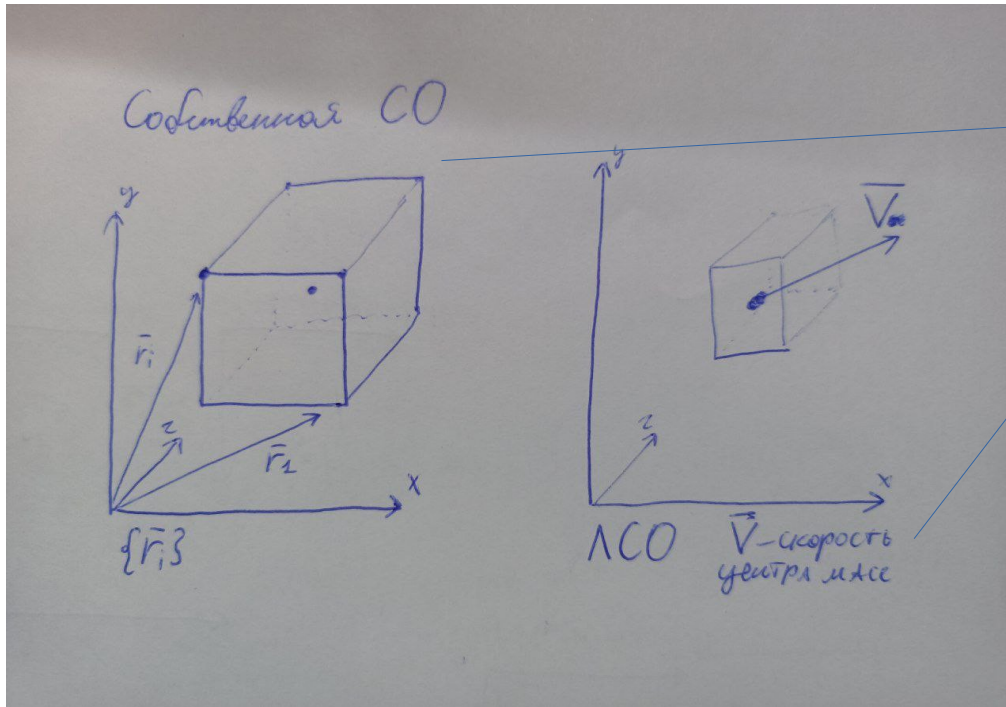
## Движение координаты центра масс объекта

- 1) Преобразования Галилея

## Dt наблюдателя (каждый тик)



# Шаг 1 — Создание объектов



$\{\vec{r}_i\}$

$\vec{V}$

```
1 await getRelativeObject(  
2   "/src/assets/wheel.gltf",  
3   new Vector3(0, 0, -1),  
4   new Vector3(0, 0, 0),  
5   new Vector3(0.1, 0.1, 0.1),  
6   new Vector3(Math.PI/2, 0, 0)  
7 ),
```

$\{\vec{r}_i\}$  — собственные координаты точек объекта

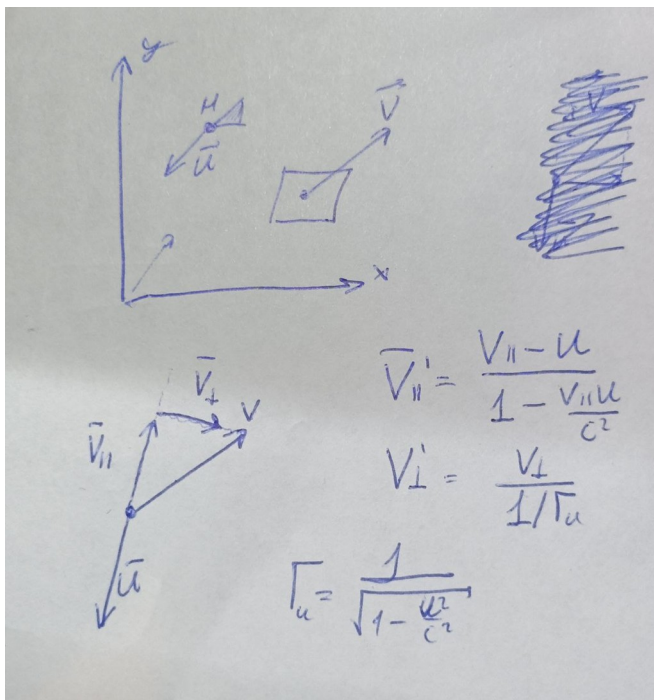
$\vec{V}$  — скорость в лабораторной системе отсчета

Остальное — позиция, масштаб и угол не  
очень на что-то влияют, опустим их

# Шаг 2 — Переход в систему отсчета наблюдателя

$$V_x' = \frac{dx'}{dt'} = \frac{dx - u dt}{dt - \frac{u}{c^2} dx} = \frac{V_x - u}{1 - \frac{u V_x}{c^2}}$$

$$V_y' = \frac{dy'}{dt'} = \frac{dy}{dt \gamma} = \frac{V_y}{\gamma}$$



Разложение вектора  $V$  по  $U$  осям

$V_{\text{порал'}}$

$V_{\text{перп'}}$

$V_{\text{перп'}} + V_{\text{порал'}}$

```

1  getRelativeVelocity(objectVelocity) {
2    const u = this.velocity.clone();
3    const v = objectVelocity.clone();
4
5    const beta = u.length();
6    const direction = u.clone().normalize();
7
8    const vParallel = direction.clone().multiplyScalar(v.dot(direction));
9    const vPerpendicular = v.clone().sub(vParallel);
10
11    const parallelSign = Math.sign(v.dot(direction));
12    const parallelMagnitude = vParallel.length();
13    const transformedParallel = direction.multiplyScalar(
14      (parallelMagnitude * parallelSign - beta) /
15      (1 - beta * parallelMagnitude * parallelSign)
16    );
17
18    const gamma = 1 / Math.sqrt(1 - beta * beta);
19    const transformedPerpendicular = vPerpendicular.multiplyScalar(1 / gamma);
20
21    return transformedParallel.add(transformedPerpendicular);
22  }

```

\*Расчет центра масс делается как и везде



# Шаг 3 — Движение

Перемещение центра масс

```
1 function update(dt, getObserverVelocity) {  
2   const localGeometry = this.object.children[0].geometry;  
3   const projGeometry = this.projObject.children[0].geometry;  
4  
5   const relVel = getObserverVelocity(this.velocity);  
6   const gamma = 1 / Math.sqrt(1 - relVel.lengthSq());  
7  
8   const dx = relVel.clone().multiplyScalar(dt);  
9   this.position.add(dx);  
10  
11   const vertices = localGeometry.attributes.position.array;  
12   for (let i = 0; i < vertices.length; i += 3) {  
13     const localPoint = new THREE.Vector3(  
14       vertices[i],  
15       vertices[i + 1],  
16       vertices[i + 2]  
17     );  
18  
19     const transformedPoint = this.lorentzTransform(  
20       dt,  
21       localPoint,  
22       relVel,  
23       gamma  
24     );  
25  
26     transformedPoint.add(this.position);  
27     projGeometry.attributes.position.setXYZ(  
28       i / 3,  
29       transformedPoint.x,  
30       transformedPoint.y,  
31       transformedPoint.z  
32     );  
33   }  
34 }  
35  
36 function lorentzTransform(dt, point, vel, gamma) {  
37   const velMod = vel.length();  
38   const direction = vel.clone().normalize();  
39  
40   const xParallel = direction.dot(point);  
41   const localDt = (dt + velMod * xParallel * gamma) / gamma;  
42  
43   const xPerpendicular = point  
44     .clone()  
45     .sub(direction.clone().multiplyScalar(xParallel));  
46   const transformedParallel = gamma * (xParallel - velMod * localDt);  
47  
48   const transformedPoint = xPerpendicular  
49     .clone()  
50     .add(direction.multiplyScalar(transformedParallel));  
51  
52   return transformedPoint;  
53 }  
54 }
```

$$\begin{cases} t' = (t - \frac{v}{c^2} x) \Gamma \\ x' = (x - v t) \Gamma \end{cases}$$

$$dt = (dt' - \frac{v}{c^2} x) \Gamma \Leftrightarrow dt' = \frac{dt + \frac{v}{c^2} x}{\Gamma}$$
$$x' = (x - v dt') \Gamma$$

Переместим центр масс, и пересчитаем все точки в систему отсчета наблюдателя из собственной

Разложим радиус вектор каждой точки на поралельную относительной скорости часть и перпендикулярную, рассчитаем отдельно и сложим.  
\*Перпендикулярная часть не изменится

# Возможности симуляции

- Импорт больших моделей из gltf файлов
- Изменение скорости наблюдателя (моментальное, не ускорение)
- Остановка времени/рендеринга
- Свободное перемещение (во время остановки рендеринга)
- Изменение скорости рендеринга (линейное изменение  $dt$  тика)
- Вывод относительной скорости, гамма фактора и линейных (по  $x, y, z$ ) размеров

- Все скорости в расчете на  $c = 1$
- Все размеры в масштабе  $1 = 299\,792\,458$  метров

\*Либо можно предполагать что скорость света в отображаемом мире сопоставима с размерами объектов

# Управление

- WASDEQ для изменения скорости наблюдателя при рендеренге(все клавиши привязаны к осям координат).
- Space для остановки рендеринга/времени.
- WASDEQ для изменения координаты во время остановки(все клавиши привязаны к осям координат).
- Z для сброса скорости наблюдателя при рендеринге.

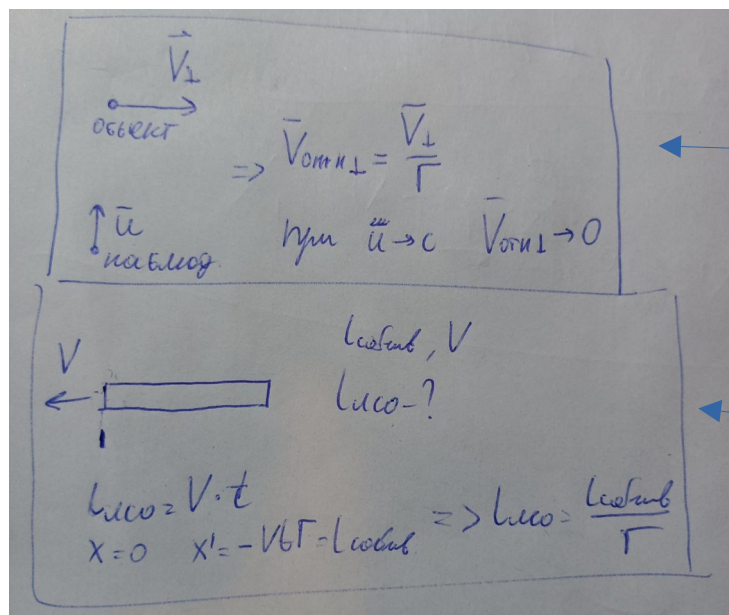


# Эксперименты

# Линейные рамеры параллелепипедов

Наблюдения:

- При движении параллелепипеда со скоростью  $0.87c$  в системе отсчёта наблюдателя он сокращается вдоль направления движения и визуально принимает форму куба. Это проявление Лоренцевого сокращения длины.
- Если перейти в систему отсчёта самого движущегося параллелепипеда, его размеры восстановятся до исходных (собственных), а неподвижный ранее параллелепипед сократится вдоль направления его движения.
- При движении объектов со скоростями, близкими к скорости света, картина для наблюдателя приобретает эффект "застывания": объекты выглядят почти неподвижными из-за релятивистского замедления времени.



$$\frac{L_{\text{contracted}}}{L_0} = 2 = \frac{1}{\sqrt{1 - \frac{v^2}{c^2}}} \Rightarrow$$

$$v = \sqrt{\frac{3}{4}} c \approx 0.866c$$

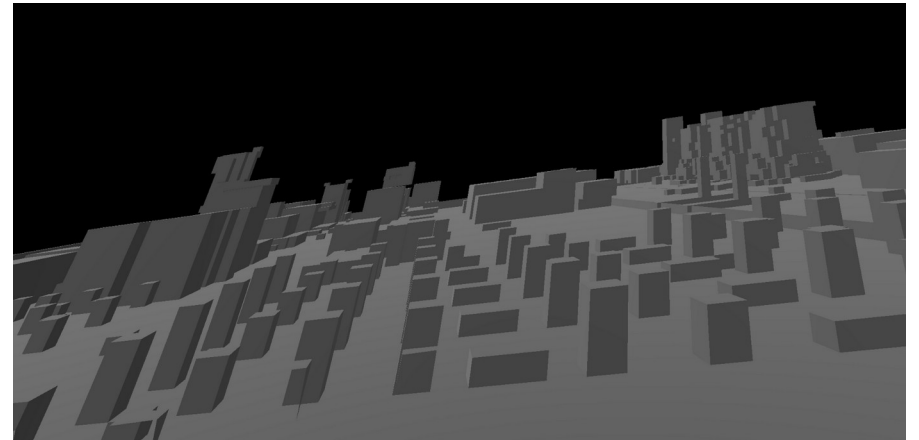
# Релятивистские гонки



Наблюдения:

- Все как и в прошлом эксперименте, но большее количество объектов с различными скоростями.
- Если у наблюдателя будет скорость не только по направлению мышин, то они начнут «вытягиваться» в сторону его движения, что прямо следует из преобразований Лоренца.

# Эксперименты 3/4



\*Все то же самое, только с более сложной геометрией объектов.

# Вращение

\*Надеюсь так можно

# Идея

Попробуем еще рассмотреть вращение объектов вокруг их центров масс, введем для этого вектор  $\omega$  собственной угловой скорости.

Предположительно, так делать можно, ведь обе системы отсчета (наблюдатель и центр масс) двигаются без ускорения, и движение твердого тела можно разложить на кручение вокруг центра масс и движения центра масс.

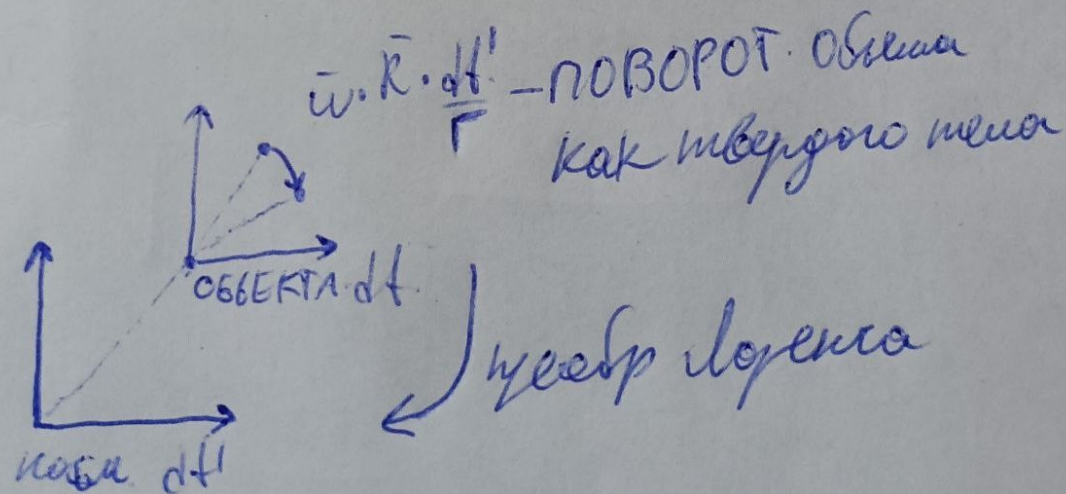


# Физические выкладки

$$dt' = (dt - \frac{V}{c^2}(x + dx))\Gamma \quad dx = dt \cdot p_{x\bar{w}} \cdot \bar{R}$$

$$dt = \frac{dt' + \frac{V}{c^2} dx \Gamma}{\Gamma(1 - \frac{V}{c^2} p_{x\bar{w}} \cdot \bar{R})}$$

$$X' = ((x + dt \cdot p_{x\bar{w}} \cdot \bar{R}) - V dt) \Gamma$$



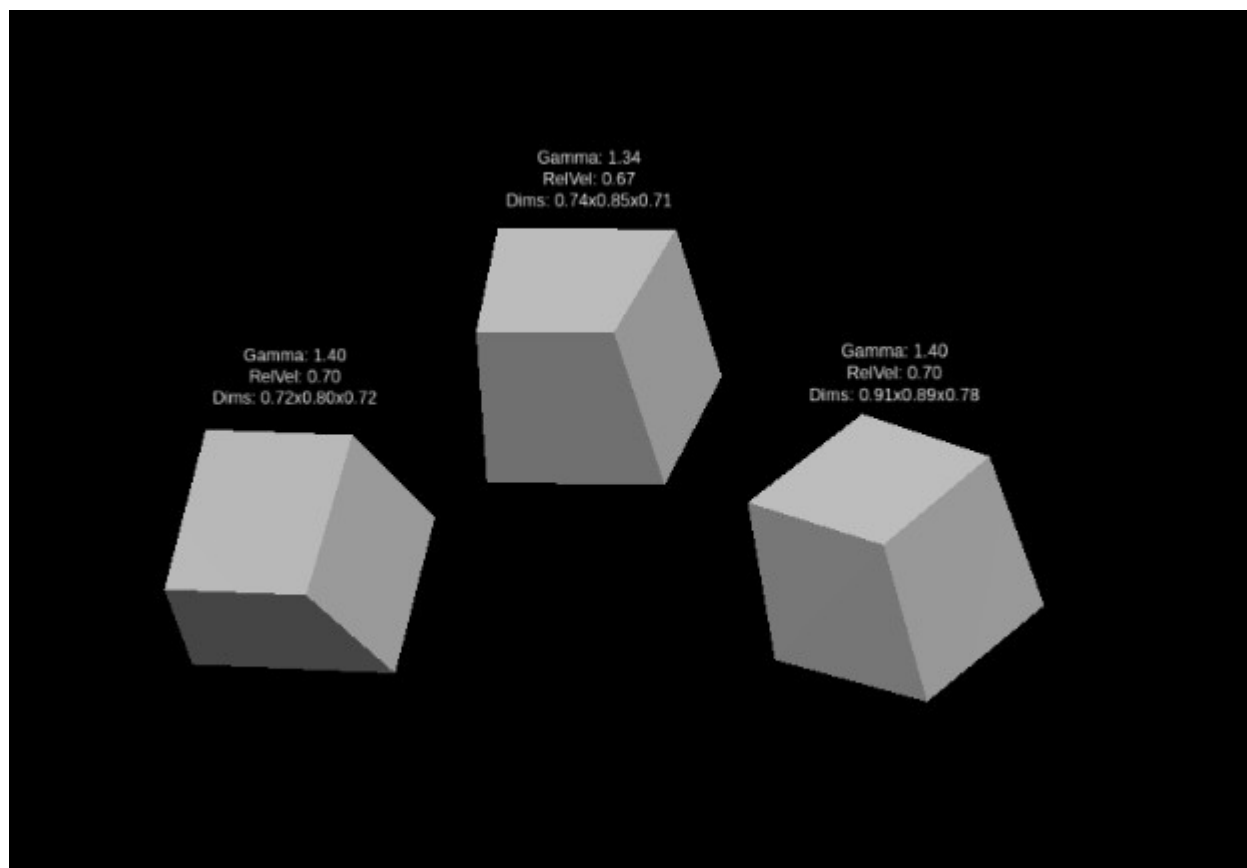
# Реализация

Усложненное нахождение  
dt, расчет нового места  
точки на его основе

Поворот как твердого тела

```
1 function update(dt, getObserverVelocity) {
2   // Геометрии (точки в системах отсчета)
3   const localGeometry = this.object.children[0].geometry;
4   const projGeometry = this.projObject.children[0].geometry;
5
6   // Значения
7   const relVel = getObserverVelocity(this.velocity);
8   const gamma = 1 / Math.sqrt(1 - relVel.lengthSq());
9
10  // Переносим центр масс
11  const dx = relVel.clone().multiplyScalar(dt);
12  this.position.add(dx);
13
14  // Преобразуем точки в глобальную систему наблюдателя с помощью преобразований Лоренца
15  const vertices = localGeometry.attributes.position.array;
16  for (let i = 0; i < vertices.length; i += 3) {
17    const localPoint = new THREE.Vector3(
18      vertices[i],
19      vertices[i + 1],
20      vertices[i + 2]
21    );
22
23    const transformedPoint = this.lorentzTransform(
24      dt,
25      localPoint,
26      relVel,
27      this.angVelocity,
28      gamma
29    );
30
31    transformedPoint.add(this.position);
32    projGeometry.attributes.position.setXYZ(
33      i / 3,
34      transformedPoint.x,
35      transformedPoint.y,
36      transformedPoint.z
37    );
38  }
39
40  // Обновляем локальное перемещение точек
41  const localDt = dt / gamma;
42  const localRotAngle = this.angVelocity.length() * localDt;
43  const localRotAxis = this.angVelocity.clone().normalize();
44
45  const rotationMatrix = new THREE.Matrix4().makeRotationAxis(
46    localRotAxis,
47    localRotAngle
48  );
49  localGeometry.applyMatrix4(rotationMatrix);
50 }
51
52 function lorentzTransform(dt, lastPoint, vel, angVel, gamma) {
53   const velMod = vel.length();
54   const direction = vel.clone().normalize();
55
56   const lastXParallel = direction.dot(lastPoint);
57   const dv = angVel.clone().multiply(lastPoint);
58   const localDt =
59     (dt + velMod * lastXParallel * gamma) / (gamma * (1 - vel.dot(dv)));
60   const point = lastPoint.clone().add(dv.clone().multiplyScalar(localDt));
61
62   const xParallel = direction.dot(point);
63   const xPerpendicular = point
64     .clone()
65     .sub(direction.clone().multiplyScalar(xParallel));
66
67   const transformedParallel = gamma * (xParallel - velMod * localDt);
68   const transformedPoint = xPerpendicular
69     .clone()
70     .add(direction.multiplyScalar(transformedParallel));
71
72   return transformedPoint;
73 }
74
```

Эсперименты можно пронаблюдать самостоятельно на сайте, пока все происходящие эффекты не смог полностью доказать, но выглядит достаточно интересно.



<https://github.com/chififid>

Спасибо за внимание!