GPU计算与工程应用

王银沼 22021071

实验环境：

CPU inter i9-10900k

GPU rtx 2080ti

操作系统 manjaro(linux)

实验内容

利用GPU加速三角形的碰撞检测，包括两个物体之间的碰撞和物体的自碰撞检测

实验原理：

使用cpu进行三角形的碰撞检测时，无论是物体间还是物体自身，都需要用到双层循环进行，如物体间碰撞检测，需要对依次判断物体a上每个三角形与物体b上每个三角形之间的碰撞情况，而每一次判断之间，都是相互独立的，所以可以利用并行算法进行加速。在cpu端，可以使用openmp库，充分利用cpu的多核，进行一定程序上的并行加速。这样的加速受限于cpu的核数，但一般cpu最多的核数也就几十个，完全无法充分实现并行算法的效率，所以使用超多核心的gpu可以显著提升效率。

实验思路：

物体间碰撞

方案一：

对于两个进行碰撞检测的物体a和b，选取三角形数量比较多的物体作为物体a。一个block负责一个物体a的三角形与所有物体b三角形的碰撞检测，所以kernel函数的block个数设为物体a三角形的个数，且为一维。一个block设置threads\_per\_block个线程，假设每个线程的id为tid，则该线程负责的物体b的三角形编号为tid+i\*threads\_per\_block，其中i表示第i次循环，共需要次数pass为：

pass=(num\_tri\_of\_b+threads\_per\_block-1)/threads\_per\_block

预先设置好每个三角形最多的碰撞个数，那么可以在kernerl启动前分配好显存，在kernel执行过程中将碰撞结果写到显存中，最后传回结果到cpu内存即可。

\_\_global\_\_ void cuTriContactDetect\_v0(tri3f\* a\_tris,tri3f\* b\_tris,

vec3f\* a\_vtxs,vec3f\* b\_vtxs,

uint a\_tris\_num,uint b\_tris\_num,

uint\* res)

{

\_\_shared\_\_ uint contacted[threads\_per\_block];

\_\_shared\_\_ uint cur\_contact\_num;

\_\_shared\_\_ Lock lock;

cur\_contact\_num=0;

int gid=threadIdx.x+blockIdx.x\*blockDim.x;

int bid=blockIdx.x;

int tid=threadIdx.x;

if (tid==0) lock.init();

contacted[tid]=0;

int pass=(b\_tris\_num+threads\_per\_block-1)/threads\_per\_block;

uint a\_tri\_x=a\_tris[bid].id(0);

uint a\_tri\_y=a\_tris[bid].id(1);

uint a\_tri\_z=a\_tris[bid].id(2);

vec3f a\_vtx\_x=a\_vtxs[a\_tri\_x];

vec3f a\_vtx\_y=a\_vtxs[a\_tri\_y];

vec3f a\_vtx\_z=a\_vtxs[a\_tri\_z];

\_\_syncthreads();

for(int i=0;i<pass;i++){

int idx=tid+i\*threads\_per\_block;

if(idx<b\_tris\_num){

uint p\_x=b\_tris[idx].id(0);

uint p\_y=b\_tris[idx].id(1);

uint p\_z=b\_tris[idx].id(2);

vec3f b\_vtx\_x=b\_vtxs[p\_x];

vec3f b\_vtx\_y=b\_vtxs[p\_y];

vec3f b\_vtx\_z=b\_vtxs[p\_z];

if(cuTriContact(a\_vtx\_x,a\_vtx\_y,a\_vtx\_z,b\_vtx\_x,b\_vtx\_y,b\_vtx\_z)){

lock.lock();

if (cur\_contact\_num<threads\_per\_block){

contacted[cur\_contact\_num]=idx;

cur\_contact\_num++;

}

lock.unlock();

}

}

\_\_syncthreads();

}

if (tid==0){

res[bid\*threads\_per\_block]=cur\_contact\_num;

}

if(tid<cur\_contact\_num && tid<threads\_per\_block-1)

res[gid+1]=contacted[tid];

}

方案二：

一个block里的每一个线程负责一个物体a的三角形与物体b所有三角形的碰撞检测。假设一个block里有threads\_per\_block个线程，那么一个block需加载这么多个相应物体a的三角形，以及每一次pass需要加载相同个数的物体b的三角形。但是每次加载的threads\_per\_block个物体b的三角形，每一个物体a的三角形都需要用到，所以可以将这部分三角形存到shared内存中。

对于threads\_per\_block个物体a和物体b的三角形，每一个物体a的三角形需要与这部分物体b的三角形进行碰撞检测，这可以通过一个循环配对实现:

第i次，三角形a(j)与三角形b((j+i)%threads\_per\_block)进行配对检测，只需要进行threads\_per\_block次即可，然后进行下一次的pass，其余部分与方案一相同。

\_\_global\_\_ void cuTriContactDetect\_v1(tri3f\* a\_tris,tri3f\* b\_tris,

vec3f\* a\_vtxs,vec3f\* b\_vtxs,

uint a\_tris\_num, uint b\_tris\_num,

uint\* res)

{

\_\_shared\_\_ uint contact\_num[threads\_per\_block];

\_\_shared\_\_ uint contacted[threads\_per\_block][8];

uint32\_t gid=threadIdx.x+blockIdx.x\*blockDim.x;

uint32\_t bid=blockIdx.x;

uint32\_t tid=threadIdx.x;

uint32\_t pass=(b\_tris\_num+threads\_per\_block-1)/threads\_per\_block;

//read a tris from global memory

tri3f a\_tri\_id=a\_tris[gid];

vec3f a\_vtx\_0=a\_vtxs[a\_tri\_id.id0()];

vec3f a\_vtx\_1=a\_vtxs[a\_tri\_id.id1()];

vec3f a\_vtx\_2=a\_vtxs[a\_tri\_id.id2()];

contact\_num[tid]=0;

\_\_syncthreads();

for(uint32\_t i=0;i<pass;i++){

uint pass\_t=i\*threads\_per\_block;

uint idx=tid+pass\_t;

\_\_shared\_\_ double cache\_b\_tris[threads\_per\_block][9];

tri3f b\_tri\_id=b\_tris[idx];

vec3f b\_vtx\_0=b\_vtxs[b\_tri\_id.id0()];

vec3f b\_vtx\_1=b\_vtxs[b\_tri\_id.id1()];

vec3f b\_vtx\_2=b\_vtxs[b\_tri\_id.id2()];

cache\_b\_tris[tid][0]=b\_vtx\_0.x;

cache\_b\_tris[tid][1]=b\_vtx\_0.y;

cache\_b\_tris[tid][2]=b\_vtx\_0.z;

cache\_b\_tris[tid][3]=b\_vtx\_1.x;

cache\_b\_tris[tid][4]=b\_vtx\_1.y;

cache\_b\_tris[tid][5]=b\_vtx\_1.z;

cache\_b\_tris[tid][6]=b\_vtx\_2.x;

cache\_b\_tris[tid][7]=b\_vtx\_2.y;

cache\_b\_tris[tid][8]=b\_vtx\_2.z;

for(uint32\_t j=0;j<threads\_per\_block;j++){

uint32\_t idx\_=(tid+j)%threads\_per\_block;

vec3f b\_vtx\_0\_{cache\_b\_tris[idx\_][0],cache\_b\_tris[idx\_][1],cache\_b\_tris[idx\_][2]};

vec3f b\_vtx\_1\_{cache\_b\_tris[idx\_][3],cache\_b\_tris[idx\_][4],cache\_b\_tris[idx\_][5]};

vec3f b\_vtx\_2\_{cache\_b\_tris[idx\_][6],cache\_b\_tris[idx\_][7],cache\_b\_tris[idx\_][8]};

if(cuTriContact(a\_vtx\_0,a\_vtx\_1,a\_vtx\_2,b\_vtx\_0\_,b\_vtx\_1\_,b\_vtx\_2\_)){

contacted[tid][contact\_num[tid]++]=pass\_t+idx\_;

}

//no need sync?

}

\_\_syncthreads();

}

res[gid\*threads\_per\_block]=contact\_num[tid];

for(uint32\_t i=0;i<contact\_num[tid];i++){

res[gid\*threads\_per\_block+i+1]=contacted[tid][i];

}

}

方案三：

方案一和方案二都是以顶点数据和顶点索引的形式存储三角形，这样子可以减少重复顶点的冗余存储，但是每次访问三角形的顶点数据时，都需要先获得三个顶点每个的索引，然后根据索引获取实际顶点数据，相当于多了一倍的全局内存访问时间，因此可以只存储顶点数据，减少访问全局内存的时间，但是相应的内存需求或增大。该方案其他思路没有变化，最后实验发现时间也基本相同。

物体自碰撞

方案一：

总体上思路与物体间碰撞方案一的思路类似，唯一不同的是一个三角形如果与所有三角形都进行碰撞检测，那么就会多一次重复测试，所以对于每个三角形，只与比它序号大的三角形进行碰撞检测即可。

\_\_global\_\_ void cuTriContactDetect\_self\_v0(tri3f\* a\_tris,vec3f\* a\_vtxs,uint a\_tris\_num,uint\* res)

{

\_\_shared\_\_ uint contact\_num;

\_\_shared\_\_ uint contacted[MAX\_CONTACT\_NUM];

\_\_shared\_\_ Lock lock;

uint32\_t gid=threadIdx.x+blockIdx.x\*blockDim.x;

uint32\_t bid=blockIdx.x;

uint32\_t tid=threadIdx.x;

// printf("kernel\n");

tri3f a\_tri\_id=a\_tris[bid];

vec3f a\_vtx\_0=a\_vtxs[a\_tri\_id.id0()];

vec3f a\_vtx\_1=a\_vtxs[a\_tri\_id.id1()];

vec3f a\_vtx\_2=a\_vtxs[a\_tri\_id.id2()];

if(tid==0){

contact\_num=0;

lock.init();//不能多次初始化

}

uint32\_t pass=(a\_tris\_num+threads\_per\_block-1)/threads\_per\_block;

\_\_syncthreads();

for(uint32\_t i=bid/threads\_per\_block;i<pass;i++){

uint32\_t idx=tid+i\*threads\_per\_block;

if(idx>bid && idx<a\_tris\_num){

tri3f a\_tri\_id\_=a\_tris[idx];

if(!cuTriAdjacent(a\_tri\_id,a\_tri\_id\_)){

vec3f a\_vtx\_0\_=a\_vtxs[a\_tri\_id\_.id0()];

vec3f a\_vtx\_1\_=a\_vtxs[a\_tri\_id\_.id1()];

vec3f a\_vtx\_2\_=a\_vtxs[a\_tri\_id\_.id2()];

if(cuTriContact(a\_vtx\_0,a\_vtx\_1,a\_vtx\_2,a\_vtx\_0\_,a\_vtx\_1\_,a\_vtx\_2\_)){

lock.lock();

if(contact\_num<MAX\_CONTACT\_NUM){

contacted[contact\_num]=idx;

contact\_num++;

}

lock.unlock();

}

}

}

\_\_syncthreads();

}

if(tid==0)

res[bid\*threads\_per\_block]=contact\_num;

if(tid<contact\_num && tid<MAX\_CONTACT\_NUM)

res[gid+1]=contacted[tid];

}

方案二：

利用bvh树进行加速。无论是利用bvh进行物体间还是物体自身碰撞检测，其实都是一样的，所以本实验只实现了旗帜自碰撞的bvh加速。普通bvh树需要在cpu使用递归建立，当然也有适用并行构建的线性bvh（lbvh），本实验因为时间原因只实现了在cpu使用递归建树。将cpu端建立的树上传到gpu端，进行并行加速，所以加速的部分主要是利用bvh树进行碰撞检测。

由于kernel函数中每一个线程负责一个三角形与整棵bvh的碰撞检测，因此如果使用递归，很容易造成kernel函数执行崩溃，故采用堆栈+遍历的方法实现递归，可以保证程序不崩溃，并且提升效率。

\_\_global\_\_ void cuTriContactDetect\_bvh\_v0(Triangle \* tris, BVHNode \* root, uint tris\_num, uint\* res)

{

uint32\_t gid=threadIdx.x+blockIdx.x\*blockDim.x;

uint contact\_num=0;

if(gid<tris\_num){

cuIterativeCD(&tris[gid],root,gid,contact\_num,res);

res[gid\*threads\_per\_block]=contact\_num;

}

}

\_\_device\_\_ void cuIterativeCD(const Triangle \*tri, const BVHNode \*node, uint gid, uint &contact\_num, uint \*res)

{

BVHNode\* stack[64];

uint32\_t stack\_pos=0;

stack[stack\_pos++]=nullptr;

Bound3 bound=tri->CuGetBound3();

const BVHNode\* \_node=node;

do{

// \_node=nullptr;//ok

if(\_node->IsLeafNode() ){

if(\_node->object\_->CuGetID()>tri->CuGetID()){

auto v\_id0=tri->CuGetTriID();

auto v\_id1=\_node->object\_->CuGetTriID();

bool is\_adj=false;

for(int i=0;i<3;i++)

for(int j=0;j<3;j++)

if(v\_id0.id(i)==v\_id1.id(j)){

is\_adj=true;

break;

}

if(!is\_adj && cuTriContact(tri->CuV0(),tri->CuV1(),tri->CuV2(),\_node->object\_->CuV0(),\_node->object\_->CuV1(),\_node->object\_->CuV2())){

res[gid\*threads\_per\_block+contact\_num+1]=\_node->object\_->CuGetID();

contact\_num++;

}

}

\_node=stack[--stack\_pos];

}

else{

auto left=\_node->left\_;

auto right=\_node->right\_;

bool traverseL=Insect(bound,left->GetBound3());

bool traverseR=Insect(bound,right->GetBound3());

\_node=(traverseL)?left:right;

if(traverseL && traverseR)

stack[stack\_pos++]=right;

}

}while(\_node!=nullptr);

}

实验结果

物体间碰撞检测（kneel-lowres数据）

方案一：

Start cd checking ...

-------------------------------

GPU Native(version 0):

GPU collision detected number is: 1083

GPU collision detected time used: 76 ms

CPU Naive:

CPU collision detected number is: 1083

CPU collision detected time used: 360 ms

-------------------------------

Finish checking: 0.65377 seconds

方案二：

Start cd checking ...

-------------------------------

GPU Native(version 1):

GPU collision detected number is: 1083

GPU collision detected time used: 93 ms

CPU Naive:

CPU collision detected number is: 1083

CPU collision detected time used: 365 ms

-------------------------------

Finish checking: 0.67871 seconds

方案三：

Start cd checking ...

-------------------------------

GPU Native(version 2):

GenerateTrisData

obj a data generate finish...

obj b data generate finish...

generate tris data finish...

GPU collision detected number is: 1083

GPU collision detected time used: 82 ms

CPU Naive:

CPU collision detected number is: 1083

CPU collision detected time used: 356 ms

-------------------------------

Finish checking: 0.63098 seconds

物体自碰撞：

run 1：

Start checking self-CD...

-------------------------------

CPU BVH:

CPU BVH build time taken: 216ms

CPU BVH collision detected number is: 1754

CPU BVH collision detection time taken: 62ms

GPU BVH:

GPU BVH build time taken: 174ms

GPU collision detected number is: 1754

GPU collision detected time used: 7 ms

GPU Naive:

GPU collision detected number is: 1754

GPU collision detected time used: 546 ms

CPU Naive:

CPU collision detected time used: 7154 ms

CPU collision detected number is: 1754

-------------------------------

Finish checking: 8.17685 seconds

run 2：

Start checking self-CD...

-------------------------------

CPU BVH:

CPU BVH build time taken: 212ms

CPU BVH collision detected number is: 1754

CPU BVH collision detection time taken: 62ms

GPU BVH:

GPU BVH build time taken: 91ms

GPU collision detected number is: 1754

GPU collision detected time used: 4 ms

GPU Naive:

GPU collision detected number is: 1754

GPU collision detected time used: 553 ms

CPU Naive:

CPU collision detected time used: 7465 ms

CPU collision detected number is: 1754

-------------------------------

Finish checking: 8.40536 seconds

run 3：

Start checking self-CD...

-------------------------------

CPU BVH:

CPU BVH build time taken: 211ms

CPU BVH collision detected number is: 1754

CPU BVH collision detection time taken: 61ms

GPU BVH:

GPU BVH build time taken: 89ms

GPU collision detected number is: 1754

GPU collision detected time used: 4 ms

GPU Naive:

GPU collision detected number is: 1754

GPU collision detected time used: 538 ms

CPU Naive:

CPU collision detected time used: 7616 ms

CPU collision detected number is: 1754

-------------------------------

Finish checking: 8.53671 seconds

实验结论