```
1. struct BezierLine
   {
    float2 CP[3];       //Control points for the line
    float2 *vertexPos; //Vertex position array to tessellate into
    int nVertices;      //Number of tessellated vertices
   };

2. __global__ void computeBezierLinePositions(int lidx, BezierLine*
       bLines, int nTessPoints)
   {
3.  int idx = threadIdx.x + blockDim.x*blockIdx.x; //Compute an
       index unique to this vertex
4.  if(idx < nTessPoints){
5.   float u = (float)idx/(float)(nTessPoints-1); //Compute u from idx
     float omu = 1.0f - u;   //Pre-compute one minus u

     float B3u[3];   //Compute quadratic Bezier coefficients
     B3u[0] = omu*omu;
     B3u[1] = 2.0f*u*omu;
     B3u[2] = u*u;

     float2 position = {0,0};  //Set position to zero
     for(int i = 0; i < 3; i++){
        //Add the contribution of the i'th control point to position
          position = position + B3u[i] * bLines[lidx].CP[i];
     }

     bLines[lidx].vertexPos[idx] = position; //Assign the value of the
          vertex position to the correct array element
     }
    }
    __global__ void computeBezierLinesCDP(BezierLine *bLines, int
        nLines)
    {
6.   int lidx = threadIdx.x + blockDim.x*blockIdx.x;  //Compute a
        unique index for each Bezier line

7.   if(lidx < nLines){
       //Compute the curvature of the line
       float curvature = length(bLines[lidx].CP[1] - 0.5f*(bLines[lidx]
                .CP[0] + bLines[lidx].CP[2]))/length(bLines[lidx].CP[2]-
               bLines[lidx].CP[0]);
       //From the curvature, compute the number of tessellation points
       bLines[lidx].nVertices = min(max((int)(curvature*16.0f),4),
       MAX_TESS_POINTS);

8.     cudaMalloc((void**)&bLines[lidx].vertexPos, bLines[lidx]
       .nVertices*sizeof(float2));
       //Call the child kernel to compute the tessellated points for
       each line
9.     computeBezierLinePositions<<<ceil((float)bLines[lidx]
       .nVertices/32.0f), 32>>>(lidx, bLines,bLines[lidx].nVertices);
   }
}
```