Launch Modes

Broadcasting

Launch a *grid of thread groups* per record, similar to a compute shader dispatch. Grid size can either be declared statically or loaded dynamically as part of the record.

If you want a dynamic grid, define it as part of the record using `SV_DispatchGrid` semantic.

```
struct Type {
    uint3 grid : SV_DispatchGrid;
    float3 someMoreData;
};
```

```
[Shader("node")]
//launch node in broadcasting mode
[NodeLaunch("broadcasting")]
//declare either a static grid of thread groups
[NodeDispatchGrid(x, y, z)]
//or the upper limits of a dynamic grid
[NodeMaxDispatchGrid(x, y, z)]
//define grid of threads per thread group
[NumThreads(x, y, z)]
void BroadcastingNode(
  uint3 gid : SV_GroupId,
  uint3 gtid : SV_GroupThreadId,
  //declare DispatchNode input record
  DispatchNodeInputRecord<Type> record,
  //declare maximum output records, where
  //n is per thread group!
  [MaxRecords(n)]
  //declare output type and target node
  NodeOutput<Type> targetNode,
  //add more outputs...
){
  //load input record of the full grid
  Type inputData = record.Get();
  //do some work...
```

Thread

Launch *one thread* per record. The GPU will try to combine a sufficient amount of records, e.g., 32, to a group, similar to a vertex or ray-generation shader. Here, group operations are not allowed.

```
[Shader("node")]
//launch node in thread mode
[NodeLaunch("thread")]
void ThreadNode(
  //declare ThreadNode input record
  ThreadNodeInputRecord<TypeA> record,
  //declare maximum output records, where
  //n is per thread!
  [MaxRecords(n)]
  //declare output type and target node
  NodeOutput<TypeB> target,
  //add more outputs...
){
  //load input record of the single thread
  TypeA inputData = record.Get();
  //maybe write two outputs per thread,
  //group outputs are forbidden in this mode!
  ThreadNodeOutputRecords<TypeB> o =
    target.GetThreadNodeOutputRecords(2);
  o.Get(0).pi = 3;
  o.Get(1).pi = 4;
  o.OutputComplete();
  //do some work...
```

Download our work graph playground here:
`https://wgpa.short.gy`

Coalescing

Launch *one thread group* for up to n records. Similar to the thread launch, but the user can specify the maximum number of records to combine, as well as the thread-group size to work on them. There is no guarantee that a node actually receives the specified number, so always use `.Count()` to the get the actual amount. Group operations are allowed.

```
[Shader("node")]
//launch node in coalescing mode
[NodeLaunch("coalescing")]
//define grid of threads per thread group
[NumThreads(x, y, z)]
void CoalescingNode(
  //declare max number of input records per
  //thread group
  [MaxRecords(n)]
  //declare GroupNode input records
  GroupNodeInputRecords<Type> records,
  uint3 gtid : SV_GroupThreadId
){
  //user can do own record-thread allocation
  int record  = gtid.x / 4;
  int channel = gtid.x % 4;
  if(record < records.Count()){
    float v =
      records.Get(record).rgba[channel];
  //do some work...
```

Record Output

Output Attributes

Define maximum number of output records per thread group (broadcasting/coalescing launch mode) or per thread (thread launch mode)
`[MaxRecords(n)]`
Instead of defining `MaxRecords(n)`, share the output record limit with some other, previously declared output.
`[MaxRecordsSharedWith("name")]`
Instead of deriving the target node from the output variable name, set target node explicitly:
`[NodeID("name")]`
Node arrays are useful to handle many nodes of same record. `[NodeID("name", idx)]` refer to a node from a node array
`[NodeArraySize(count)]` set array size
`[UnboundedSparseNodes]` no array size limit
`[AllowSparseNodes]` allow some node array indices to be invalid

Empty Outputs

The simplest way to create work is to output records without any data. Output limits per thread or group do not apply to empty outputs.

```
[NodeLaunch("broadcasting")]
NodeDispatchGrid(1, 1, 1)]
[NumThreads(32, 1, 1)]
void node(
  //maximum output must be declared
  //but value may exceed limits for non-empty outputs
  [MaxRecords(1024)]
  EmptyNodeOutput targetNode
){
  //outputs 512 records for the whole group, 512 remain available...
  targetNode.GroupIncrementOutputCount(512);
  //outputs 64 records per thread or 32 · 64 = 512 for the whole group
  targetNode.ThreadIncrementOutputCount(64);
```

Data Outputs

Records with data have slightly different syntax than empty outputs.

```
[NumThreads(32, 1, 1)]
void node(
  uint gtid : SV_GroupThreadId
  //maximum must not exceed limits
  [MaxRecords(128)]
  NodeOutput<TypeA> targetNodeA
  [MaxRecords(64)]
  NodeOutput<TypeB> targetNodeB
){
  //output 64 records for the whole thread group for A, 64 remain...
  GroupNodeOutputRecords<TypeA> outputA =
    targetNodeA.GetGroupNodeOutputRecords(64);
  //cooperatively write to this record as a whole thread group
  outputA.Get(2 * gtid + 0) = someData0;
  outputA.Get(2 * gtid + 1) = someData1;
  //finish output for target node A
  outputA.OutputComplete();

  //output 2 B records per thread or 32 · 2 = 64 for the whole group
  ThreadNodeOutputRecords<TypeB> outputB =
    targetNodeB.GetThreadNodeOutputRecords(2);
  //a single thread writes to its records
  outputB.Get(0) = someMoreData0;
  outputB.Get(1) = someMoreData1;
  outputB.OutputComplete();
```

Limits

maximum (non empty) records per output per thread group: 256
  per thread-launched thread: 8 = 256/32
maximum total size of outputs per thread group: 32KB
  per thread-launched thread: 128B