Name: Henry Jiang
Contact: @atrxi :blender.org
Email: henryyjiang42@gmail.com

Summary:

This task involves implementing enhanced loop-editing operations in edit-mode to provide additional options to modelers and accelerate their workflow. These operations will be:

1. **"Clone" Support for Edge Slide:** An option for the Edge Slide tool which, when enabled, duplicates the corresponding edges first and then slides them. See RCS#2rcbbc, RCS#QDbbbc.
2. **Edge Flow:** Adjusts the edge loop via spline interpolation such that it respects the flow of the surrounding geometry. See RCS#vddbbc.
3. **Loop Cut Curvature Preservation:** Option added to loop cutting that when enabled places loop vertices on the curvature of the surrounding geometry with spline interpolation rather than flat linear interpolation. See RCS#fSdbbc

Expected Project Size: 350 Hours

Benefits:

Each of these additions will offer functionality that accelerates the workflow for general tasks and addresses workflow friction in Blender's current available options.

"Clone" Support for Edge Slide replicates existing behavior from Modo and C4D that has been requested by many users, and offers powerful options for hard modeling. Currently the best available option is `Shift+D` to duplicate, cancel with `RMB`, then `G, G` to edge slide. This behavior can also be messy when multiple edges are selected across complex geometry. Clone support would cut out the unintuitive step of duplicating the edge and canceling the movement and reduce this to simply `G+G.`

Edge Flow is a popular addon that many modelers believe to be essential, marking a significant gap in default Blender capabilities. With the popularity of the addon, it makes sense to have it available by default for users for convenience. Edge Flow solves the problem of adjusting edge loops to better match the flow of surrounding geometry, and can also create linear edge loops to straighten out

geometry. Default Blender behavior would involve manually adjusting individual edge loops to achieve the same shape. A native implementation would not only be much faster and have more seamless integration, but would also have access to normals, CustomData layers, and loop data inaccessibly via Python extensions.

Loop Cut Curvature Preservation addresses friction points in organic curvature modeling for adding detail without altering the shape. Typically, a subdivision is inserted via linear interpolation which must then be manually adjusted to fit the curvature of the object. This feature would replicate the default behavior of Maya's Insert Edge Loop and Modo's Loop Slice. A native implementation in Blender would offer benefits over an extension based implementation due to access to vertex normals and the smoothness parameter and existing spherical interpolation methods at the C level.

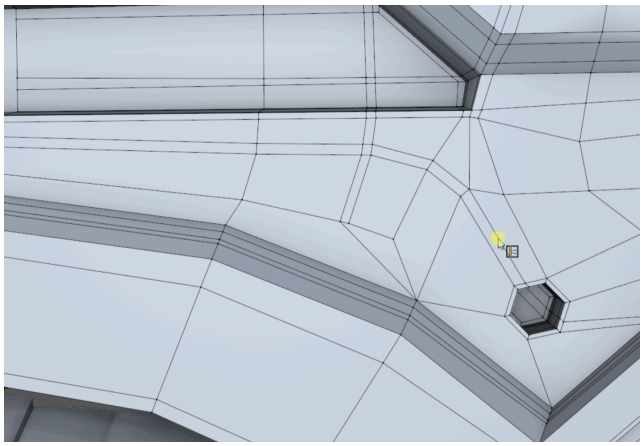
Deliverables:

1. **Clone Support for Edge Slide** - A boolean toggle will be added to `MESH_QT_edge_slide` in `transform_mode_edge_slide.c`, displaying as a checkbox in the options UI panel. This operation duplicates the selected edges in place and then slides them according to cursor tracked position. The new edges will be connected to the original edges by faces and will slide along the existing neighborhood independently, leaving the original edges untouched. This registers as a single undo step. Sliding behavior should be consistent with Cinema 4D and Modo implementations.
2. **Edge Flow** - New native BMesh [2] operator `MESH_QT_set_edge_flow`, porting and integrating Benjamin Sauder's "Edge Flow" add-on into the C level codebase. The operator has two modes. Flow mode uses Catmull-Rom spline interpolation [3] to reposition each vertex in a selected edge loop to the surrounding adjacent geometry to curve. A tension weight function for this mode should be present. The second linear mode should allow straight distribution of selected loops between endpoints. The operator should handle open and closed loops and respect custom normals and UV seams via BMesh CustomData access. The operator should be accessible from the edge menu in edit mode and can be searched for with F3 and is custom keybindable.
3. **Loop Cut Curvature Preservation** - A boolean toggle `use_curvature_preservation` will be added to `MESH_OT_loopcut` in `editmesh_loopcut.cc` to be toggled in the options UI panel. Curvature preservation will use Hermite spline interpolation rather than linear interpolation

for default and spherical interpolation for smoothing to place the new loop cut. A tension parameter will be implemented as well to control the amount of curvature application, and much of the code from Edge Flow can be adapted rather directly. The operator should apply properly to multiple cuts as well.

Proposed Solutions:

“Clone” Support for Edge Slide:



See RCS#2rcbbc for video demonstration.

The implementation will focus on an edge duplication step before the edge slide that clones the edge in place and slides it along existing geometry. Endpoints of the clone will connect back to the original boundary, the originals will be deselected, and the rest of the operation will continue as usual. The entire operation will be encompassed into one undo step. As of now, the 2 anticipated files that will need editing are:

1. `source/blender/editors/transform/transform_mode_edge_slide.cc`
2. `source/blender/editors/transform/transform_ops.cc`

Current Logic:

When `G, G` is pressed, `TRANSFORM_OT_edge_slide` calls `initEdgeSlide()` in `transform_mode_edge_slide.cc` to read RNA properties from the operator. `initEdgeSlide_ex()` allocates `EdgeSlideParams` to `t->custom.mode.data` and calls `createEdgeSlideVerts()` for all `TransDataContainers`.

```
FOREACH_TRANS_DATA_CONTAINER (t, tc) {
   EdgeSlideData *sld = static_cast<EdgeSlideData *>(tc->custom.mode.data);

   if (sld == nullptr) {
     continue;
   }

   for (TransDataEdgeSlideVert &sv : sld->sv) {
     edge_slide_apply_elem(
         sv, perc, curr_length_fac, curr_side_unclamp, use_clamp, use_even, use_flip,
sv.td->loc);
   }
 }
```

`transform_mesh_edge_slide_data_create()` then walks BMesh selection and creates an array stored in `EdgeSlideData::sv` for all the vertices that need to be slid along the existing edges. Vertex position for slide factor s is computed as such:

$$p_{new} = p_{old} + |s| \cdot d_s$$

Where s lies between [-1,1] and selects `dir_side[0]` or `dir_side[1]` based on sign and d_s is the full displacement vector.

```
FOREACH_TRANS_DATA_CONTAINER (t, tc) {
 EdgeSlideData *sld = static_cast<EdgeSlideData *>(tc->custom.mode.data);

 if (sld == nullptr) {
   continue;
 }

 for (TransDataEdgeSlideVert &sv : sld->sv) {
   edge_slide_apply_elem(
       sv, perc, curr_length_fac, curr_side_unclamp, use_clamp, use_even, use_flip,
sv.td->loc);
 }
}
```

Each modal event is handled through `handleEventEdgeSlide()` for `E` (Even Mode), `F`(Flip), `C`(Clamp) inputs. Mouse movement is handled by `calcEdgeSlideCustomPoints()`.

Implementation Overview:

Our duplication will take place in `initEdgeSlide()` in `transform_mode_edge_slide.cc` before the call to `initEdgeSlide_ex()` because `createEdgeSlideVerts` operates on the selection in the BMesh. The operation will duplicate the selected edges in place, connect endpoints of the clone to the original edges, unselect the original edges and then continue with our new selection.

Firstly, we'll need a `use_clone` flag added to the properties to enable the selection of the behavior from the UI panel later on. We can also implement a new method `edge_slide_duplicate_selection()` to handle the behavior of making a copy of the original edge and reconnecting to original geometry. This can likely be handled with the BMesh `duplicate` operation.

```
prop = RNA_struct_find_property(op->ptr, "use_clone");
bool use_clone;

if (use_clone) {
 FOREACH_TRANS_DATA_CONTAINER (t, tc) {
   duplicate_selection(t, tc);
 }
}
initEdgeSlide_ex(*args);
```

edge_slide_duplicate_selection() Pseudocode:

```
for each selected edge e:
 Tag original
Duplicate selection
Collect original and duplicate edge sets
Use dir_side[0] and dir_side[1] to decide which edges to bridge to
Bridge old selection to new with quad faces
Switch selection to clone only
```

The clone will be coincident with originals so the initial position is identical to before the operation. As the user slides along the edge, the clone separates and quads emerge to fill the gap. This replicates clone behavior in C4D and Modo. To determine

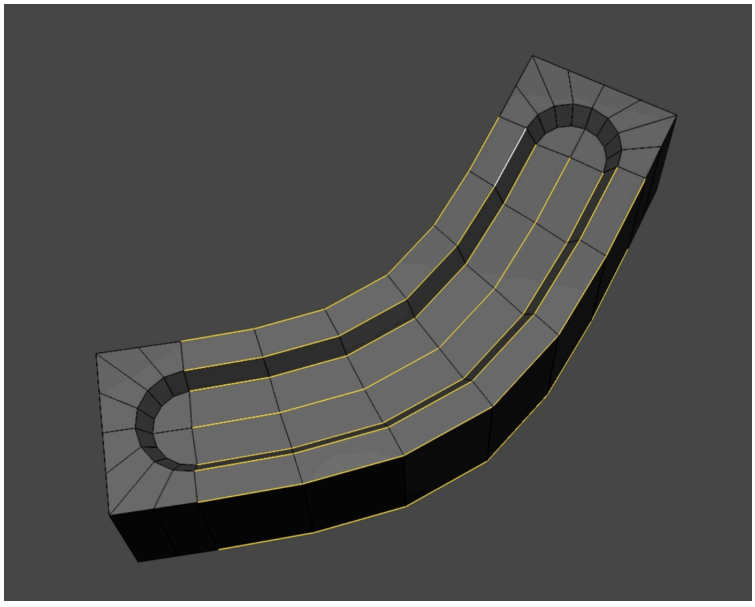
the edges in closed geometry to remove and replace with connections to the original edge selection, we will leverage the logic already present from createEdgeSlideVerts and `transform_mesh_edge_slide_data_create` reading `dir_side[0]` and `dir_side[1]`.

Other requirements for implementation would be to condensing the operation into a single undo step for `EDBM_update()` calls. The `BMO_op_initf`, `BMO_op_exec`, `BMO_op_finish` pattern may be an effective implementation here to prevent an intermediate `EDBM_update()` call. RNA property registration in `transform_ops.cc` is also necessary for the `use_clone` boolean so that it appears at the top of the options UI panel.

Edge Cases

Additional edge cases are likely for interaction with Even Mode, Mirror Modifier, and multiple loops/objects. Even Mode will have duplicated vertices that start coincident with the originals and will slide in a fixed manner rather than interpolated. Mirror modifier will probably be handled by the modifier with `TransDataContainer` already having `use_mirror_axis` flags, but it would be important to test this extensively. It would also be ideal to test selections of multiple loops at once and across disconnected geometry, although `FOREACH_TRANS_DATA_CONTAINER` does address this behavior inherently, iterating per object container in `initEdgeSlide()`.

Edge Flow:



See EdgeFlow for video demonstration.

Edge Flow is a popular add-on by Benjamin Sauder that repositions an edge loop's vertices via spline interpolation accounting for surrounding geometry. This implementation will involve a linear mode and a flow mode. Much of the process will involve adapting existing logic from the add-on to C level code and improving integration with CustomData at the C level. 4 files are anticipated to be modified at minimum.

Relevant Files:

1. `source/blender/editors/mesh/editmesh_edge_flow.cc [NEW]`
2. `source/blender/editors/mesh/mesh_ops.cc`
3. `source/blender/editors/mesh/editmesh_select.cc`
4. `source/blender/bmesh/intern/bmesh_interp.cc`

BMWalker edge loop traversal is in `editmesh_select.cc`. For this implementation a new `MESH_OT_set_edge_flow` operator needs to be added to `editmesh_edge_flow.cc`. `Mesh_ops.cc` also needs to be updated to register this new operator, and `mesh_interp.cc` is needed for referencing CustomData.

Current Logic:

Blender does not possess a native operator that performs the same function as Edge Flow. For the subsequent section, Edge Flow's Python code will be analyzed and equivalent code snippets written in C will be provided for C level code in the Blender codebase.

- Edge Flow's `walk_edge_loop()` function in `util.py` has a C equivalent in editmesh_select.cc called `BMW_EDGELOOP`. These functions both walk along the parallel edge ring. Edge Flow lacks handling of CustomData present in `BM_loop_interp_from_face()` from `bmesh_interp.cc`:

```
void BM_loop_interp_from_face(
   BMesh *bm, BMLoop *l_dst, const BMFace *f_src,
   const bool do_vertex, const bool do_multires);
```

This is a key issue that should be addressed in the implementation.

Implementation Overview:

The first step in Edge Flow's implementation of Flow Mode is to collect each selected edge in the loop, and the rings for per edge traversal.

```
# edgeloop.py: set_flow()
for loop in edge.link_loops:
    ring1 = loop.link_loop_next.link_loop_next
    ring2 = loop.link_loop_radial_prev.link_loop_prev.link_loop_prev
```

A C equivalent implementation would use BMLoop for the same effect:

```
BM_ITER_ELEM(loop, &liter, edge, BM_LOOPS_OF_EDGE) {
   BMLoop *ring1 = loop->next->next;
   BMLoop *ring2 = loop->radial_prev->prev->prev;

   BMVert *center;
   BMVert *p2 = ring1->v;
   BMVert *p3 = ring2->radial_next->v;
}
```

Flow Mode:

The addon uses Kochanek-Bartels Hermite spline interpolation, implemented in hermite_3d() in `interpolate.py`. The formula is as follows:

$$m_0 = ((p_2 - p_1)(1 + bias)(1 - tension) + (p_3 - p_2)(1 - bias)(1 - tension))/2$$

$$m_1 = ((p_3 - p_2)(1 + bias)(1 - tension) + (p_4 - p_3)(1 - bias)(1 - tension))/2$$

Bias determines a bias to either direction, with 0 defaulting to symmetric and reduces the standard Catmull-Rom tangent form [3].

$$m_0 = (p_3 - p_1)(1 - tension)/2$$

$$m_1 = (p_4 - p_2)(1 - tension)/2$$

The full Hermite spline interpolation equation is as follows, where \mu represents a parameter 0.5 between 2 given points for interpolation. [4]

$$p(\mu) = h_{00}(\mu) \cdot p_2 + h_{10}(\mu) \cdot m_0 + h_{01}(\mu) \cdot p_4 + h_{11}(\mu) \cdot m_1$$

The hermite basis functions at parameter \mu:

$$h_{00}(\mu) = 2\mu^3 - 3\mu^2 + 1$$

$$h_{01}(\mu) = \mu^3 - 2\mu^2 + \mu$$

$$h_{00}(\mu) = -2\mu^3 - 3\mu^2$$

$$h_{00}(\mu) = \mu^3 - \mu^2$$

Edge Flow then normalizes control point distances and computes tangents. Tension values are passed in from the RNA property ranging from [-500, 500], which determines the tension of the Catmull-Rom curve.

The relevant equations for relative projection are:

$$s = (v - p_1) \cdot d$$

$$v_{new} = p_1 + s \cdot d$$

And for even spacing:

$$v_{new} = p_1 + (i/n) \cdot (p_2 - p_1)$$

Edge Flow then iteratively relaxes the loop by passing the output positions in as the next iteration's control points. A C level implementation would use the same method of iterative interpolation. Basic pseudocode is as follows:

```
Normalize control arm lengths
Kochanek-Bartels tangent calculation
Hermite spline interpolation
For i in iterations:
 Apply flow to selected edges
```

Kochanek-Bartels Hermite calculation in C:

```c
  float m0[3], m1[3];
  sub_v3_v3v3(m0, p3, np1);
  mul_v3_fl(m0, (1.0f - tension) * 0.5f);
  sub_v3_v3v3(m1, np4, p2);
  mul_v3_fl(m1, (1.0f - tension) * 0.5f);

  const float mu = 0.5f;
  const float mu2 = mu * mu, mu3 = mu2 * mu;
  const float h00 = 2*mu3 - 3*mu2 + 1;
  const float h10 = mu3 - 2*mu2 + mu;
  const float h01 = -2*mu3 + 3*mu2;
  const float h11 = mu3 - mu2;

  r_co[0] = h00*p2[0] + h10*m0[0] + h01*p3[0] + h11*m1[0];
```

```
   r_co[1] = h00*p2[1] + h10*m0[1] + h01*p3[1] + h11*m1[1];
   r_co[2] = h00*p2[2] + h10*m0[2] + h01*p3[2] + h11*m1[2];
```

Possible flow application code:

```
static void edge_flow_apply_flow(BMesh *bm,
                                 BMEdge **loop_edges)
{
   for (int i = 0; i < iters; i++) {
       for (int j = 0; j < loop_len; j++) {
           BMEdge *edge = loop_edges[j];
           BMVert *center = get_center_vert(edge);
           calc_hermite_position(;
           copy_v3_v3(center->co, new_co);
       }
   }
}
```

Linear Mode:

Linear mode is implemented in Edge Flow in the method `set_linear()` in `edgeloop.py`. It has two submodes for either relative projection or even spacing. Endpoint detection finds the vertex in the first edge not shared with the second edge and likewise for the last edge not shared by the second-last edge. This can be translated directly to C with BMEdge and BMVert based calculations.

Possible implementation:

```
for (int i = 0; i < loop_len - 1; i++) {
       BMVert *v

       if (even_spacing) {
           madd_v3_v3v3fl();
       } else {
           float proj[3];
           sub_v3_v3v3();
           float scalar = dot_v3v3());
           madd_v3_v3v3fl();
       }
       last = v;
   }
```

Edge Flow has a mix property on distribution [0,1] which blends the original and computed positions as well with linear interpolation, in C this should just be possible with interp_v3_v3v3() applied at the end after computation.

CustomData and RNA Properties:

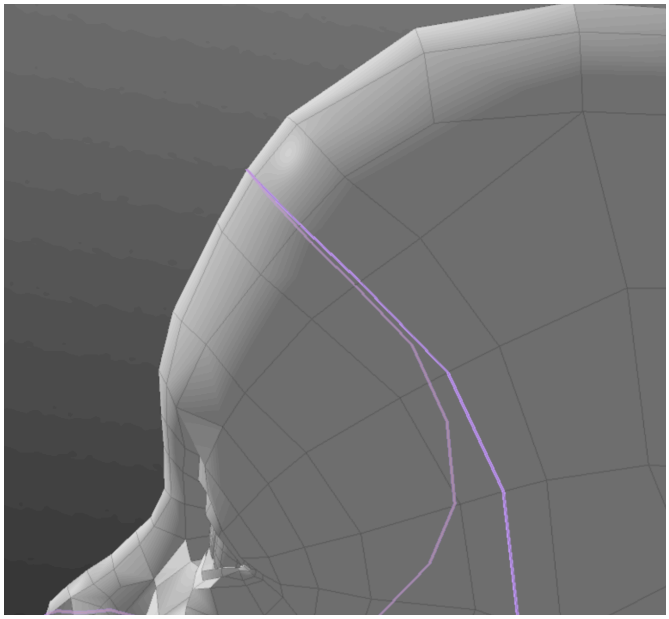
After interpolation is complete, CustomData should be refreshed automatically with `BM_loop_interp_from_face()` as mentioned prior. This can be done simply by iterating across all moved vertices and reinterpolating [1].

The operator level implementation will follow other `MESH_OT` operators. RNA property registration, `mesh_ops.cc` registration, `EDBM_update` calling, and exec/poll functions should follow the implementation of those.

Edge Cases:

Edge Flow does have documented edge cases that it handles. Specifically, n-gon adjacent faces that are not quads require explicit handling. In Edge Flow, this is handled in a separate util.py method called `walk_ngon().` Boundary edges are also handled by special ghost control points, and a minimum angle threshold ensures rings contribute less when close to linear with the ring edge. These will all be additional methods implemented. Special behavior is also likely needed for disconnected edge loops selected at the same time.

Loop Cut Curvature Preservation



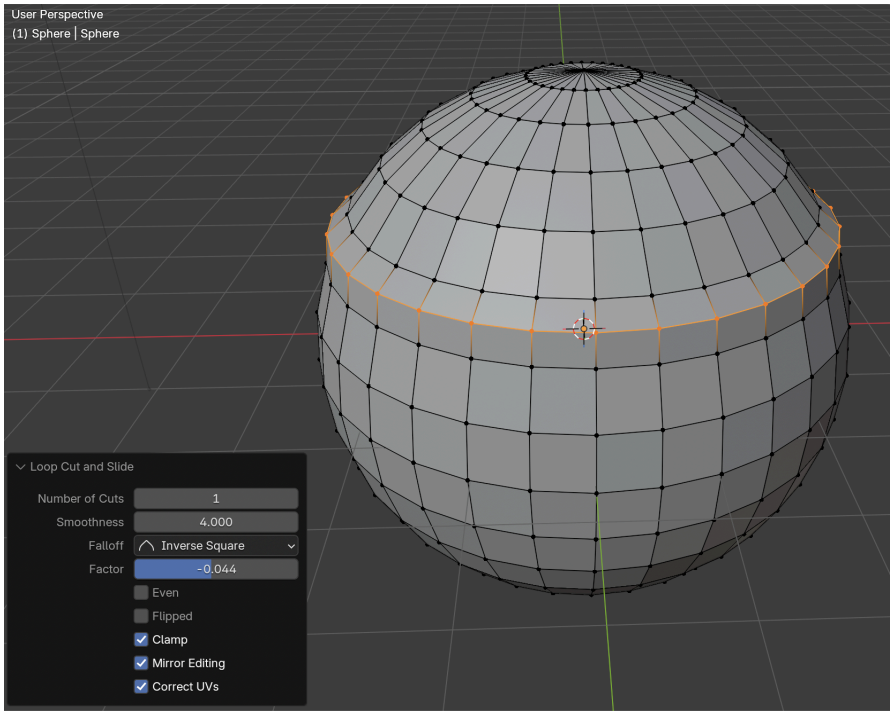
See RCS#fSdbbc for video demonstration.

The implementation will focus on implementing a boolean toggle `use_curvature_preservation` that enables spherical interpolation for loop cutting, offering more effective usage when increasing the detail of curved objects during modeling. Behavior should replicate the implementations in Maya and Modo. Currently, the only file expected to need modification is editmesh_loopcut.cc.

Current Logic:

When `Ctrl+R` is registered as input, `Mesh_QT_loopcut` is triggered and checks `VIEW3D_GGT_mesh_preselect_edgering` and `is_interactive`. If we are in interactive mode, we proceed with a modal cursor handler. Preselected edging and script-based exec modes are not relevant for the scope of our implementation.

`ringsel_finish()` calls `BM_mesh_esubdivide()` on confirmation by the user, which then computes new vertex positions in `bmo_subdivide_edges_exec()`. When use_smooth is enabled, a normal-guided spherical interpolation is calculated by `interp_slerp_co_no_v3()` which is then blended to flat linear interpolation position using the smooth weight as a factor. This behavior offsets the new edges by a certain amount along their normals; this does not automatically preserve existing curvature, as demonstrated below.



The current logic for smoothness calculation is the following:

```
creflect_v3_v3v3(no_reflect, v_a->no, no_dir);
interp_slerp_co_no_v3(v_a->co, v_a->no, v_b->co, no_reflect, no_dir, perc, co_a);

reflect_v3_v3v3(no_reflect, v_b->no, no_dir);
interp_slerp_co_no_v3(v_a->co, no_reflect, v_b->co, v_b->no, no_dir, perc, co_b);

interp_v3_v3v3(co, co_a, co_b, perc);
```

In equation form this is:

$$c_{new} = c_{sphere} + slerp(v_a, v_b, perc) \cdot lerp(|v_a|, |v_b|, perc)$$

To adapt to preserve curvature, the smoothing parameter would need to be repressed when use_preserve_curvature is enabled.

Proposed Implementation:

First, to define the new RNA boolean to show up in the options panel:

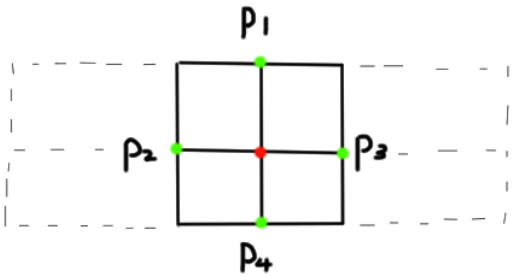
```
prop = RNA_def_boolean(ot->srna,
                       "use_curvature_preservation",
                       false,
                       "Curvature Preservation",;
RNA_def_property_flag(prop, PROP_SKIP_SAVE);
```

This implementation shouldn't need to make any direct changes to any methods in bm_esubdivide.cc, and the math for curvature preservation should be possible with existing methods. We can add in a new tension parameter to be from [0,1] distribution and default to 1.0f if use_curvature_preservation is enabled to pass into a method after BM_mesh_esubdivide() at ringsel_finish(). Smoothness should default to 0.0f if our boolean is enabled to prevent it from influencing the curvature.

```
/* In rinsel_finish() */
BM_mesh_esubdivide(em->bm, BM_ELEM_SELECT, 0.0f, smooth_falloff,
                   false, 0.0f, 0.0f, cuts, seltype,
                   SUBD_CORNER_PATH, 0, true, use_only_quads, 0);

if (use_preserve_curvature) {
   loopcut_apply_curvature(em->bm, curvature_tension);
}
```

For applying curvature preservation, the new subdivision vertices need to be perturbed into position by Hermite spline interpolation, similar to the usage in Edge Flow above. We can take the newly created vertices in the loop cut generated by linear interpolation and set the four control points as the original vertices that the loop cut vertices lie between. The points p1, p2, p3, and p4 in relation to a specific vertex are shown below.



Spline interpolation would be implemented very similarly to Edge Flow implementations. It would place the new vertices in the midpoint of neighboring ring vertices and be corrected proportionally along the curvature. In the same manner as Edge Flow, `CustomData` can be refreshed after the interpolation, recomputing normals and other information with `BM_face_interp_from_face()`. Much code from Edge Flow can be adapted to this implementation, especially for spline interpolation and edge case handling. The pseudocode for such an implementation is as follows:

```
Loopcut_apply_curvature(*bm, tension):
 Iterate over verts tagged by esubdivide:
  Find loop of vert, find 4 control points
  Catmull-Rom tangent computation
  Hermite spline interpolation with \mu param of 0.5

 Refresh normals
```

Edge Cases

The most likely edge case is near flat or completely flat geometry, so a check should be in place for that to default back to linear interpolation. Similar to Edge Flow, endpoint vertices on open loops will require special behavior to handle. Similar behavior might be necessary for very sharp edges as well. Multicut would also likely warrant extensive testing.

Timeline:

Phase 1: Weeks 1-3 ~80 Hrs

- Loop Cut Curve Preservation: Study `interp_slerp_co_no_v3` spherical interpolation usage and analyze how to adapt the smoothness parameter. Begin implementation.
- Edge Clone: Study `transform_mode_edge_slide`.c and understand `EdgeSlideData` and `TransDataEdgeSlideVert`. Begin detailed documentation about how to adapt to cloning and how to implement seamlessly without interfering with existing implementation.

Phase 2: Weeks 4-6 ~100 Hrs

- Loop Cut Curve Preservation: Add RNA property, finish implementation and conduct testing against edge cases and multicut and ensure proper blending is applied.
- Edge Clone: Begin implementation of edge duplication and sliding behavior code implementation. Wrap into a single undo step with `EDBM_update` calls and add clone boolean to RNA properties.
- Edge Flow: Study Sauder's EdgeFlow addon source, mapping operations to BMesh C equivalent. Begin work on basic code implementation, setting up overall framework.

Phase 3: Weeks 7-9 ~90 Hrs

- Edge Clone: Check functionality for multiple selected loops and conduct extensive testing across modifiers, modes, and edge cases.
- Edge Flow: Begin implementation of linear mode and flow mode, adding RNA property to UI menu.

Phase 4: Weeks 9-12 ~80 Hrs

- Edge Flow: Finalize base functionality implementation. Implement `CustomData` preservation and conduct extensive testing across edge cases.
- Address remaining bugs and edge cases, improve overall code organization and work on documentation and demonstration videos and any remaining work.

About Me:

I'm an undergraduate senior at Georgia Institute of Technology studying computer science and graduating in May of this year. My concentrations are computational media and artificial intelligence, and I have a strong background in computer graphics, mathematics, Python, and C++ from my coursework and projects.

I've contributed with two published Blender extensions with over 5,000 downloads total so far, the most notable and relevant one being an advanced topology-based selection propagation tool for Edit Mode. I've been using Blender as a 3D modeler, rigger, and animator for several years and have a strong understanding of a majority of the modules. I'm particularly intrigued by this project for its difficulty and the challenge it poses, and I'm driven to put myself out there to demonstrate my skills and produce something brilliant that other people would not attempt. I'm eager to contribute to Blender in particular for how it has enabled me to pursue my passions in animation and the incredible community around it.

My PRs:
https://projects.blender.org/blender/blender/pulls/156498
https://projects.blender.org/blender/blender/pulls/156618

My Extensions:
https://extensions.blender.org/author/65907/

Citations:

[1] Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., & Levy, B. (2010). *Polygon Mesh Processing.* A K Peters/CRC Press.
[2] Botsch, M., Steinberg, S., Bischoff, S., & Kobbelt, L. (2002). "OpenMesh: A Generic and Efficient Polygon Mesh Data Structure." *OpenSG Symposium 2002.*
[3] Catmull, E. & Rom, R. (1974). "A Class of Local Interpolating Splines." In *Computer Aided Geometric Design*, 317–326. Academic Press.
[4] Yuksel, C., Schaefer, S., & Keyser, J. (2011). "Parameterization and Applications of Catmull-Rom Curves." *Computer-Aided Design*, 43(7), 747–755.