In[∘]:=
```
SetDirectory["C:\\drorbn\\AcademicPensieve\\Projects\\HigherRank\\DunfieldKnots"];
Once[<< KnotTheory`];
<< ../Rot.m
T3 = T1 T2;
```

```
C:\drorbn\AcademicPensieve\Projects\KnotTheory\KnotTheory

Loading KnotTheory` version of February 2, 2020, 10:53:45.2097.
Read more at http://katlas.org/wiki/KnotTheory.

Loading Rot.m from http://drorbn.net/AP/Projects/HigherRank to compute rotation numbers.
```

In[∘]:=
```
CCF[ℰ_] := ExpandDenominator@ExpandNumerator@Together[ℰ];
CCF[ℰ_] := Factor[ℰ];
CF[ℰ_List] := CF /@ ℰ;
CF[ℰ_] := Module[{vs = Cases[ℰ, (x | p | π | g)__, ∞] ⋃ {x, p, ϵ}, ps, c},
   Total[CoefficientRules[Expand[ℰ], vs] /. (ps_ → c_) ↦ CCF[c] (Times @@ vs^ps)] ];
```

Data

(from Talks/Beijing-2407/theta.nb)

In[∘]:=
```
R1[1, i_, j_] = CF[
   1 / 2 - T3 g1ji g2ji - g3ii + g2jj g3ii + T1 (T3 - 1) g1ji g3ji +
    T2 (T3 - 1) g2ji g3ji - T2 g2ji g3jj + (g1jj g2ii + (T3 - 1) g1jj g2ji -
       T1 g1ii g2jj - g1jj g3ii - T1 (T3 - 1) g1jj g3ji + T1 g1ii g3jj) / (T1 - 1)];
```

In[∘]:=
```
R1[-1, i_, j_] = CF[
   -1 / 2 - T1^-1 g1ji g2ii - (1 - T1^-1 - T2^-1) g1ji g2ji - g1jj g2ji - g1ji g2jj + g3ii +
    T1^-1 g1ji g3ii - (1 - T2^-1) g2ji g3ii - g2jj g3ii + (1 - T3^-1) g1ji g3ji - (1 - T3^-1) g2ii g3ji +
    (2 - T2^-1) (1 - T3^-1) g2ji g3ji + (1 - T3^-1) g2jj g3ji + g1ji g3jj + g2ji g3jj + (T1 (1 - T2^-1) g1ii g2ji -
       g1jj g2ii + T1 g1ii g2jj + g1jj g3ii - T2^-1 (T3 - 1) g1ii g3ji - T1 g1ii g3jj) / (T1 - 1)];
```

In[∘]:=
```
θ[{1, i0_, j0_}, {1, i1_, j1_}] =
  -T1 (T3 - 1) g1,j1,i0 g2,i1,i0 g3,j0,i1 + (T3 - 1) g1,j1,j0 g2,i1,i0 g3,j0,i1 +
   T1 (T3 - 1) g1,j1,i0 g2,j1,i0 g3,j0,i1 - (T3 - 1) g1,j1,j0 g2,j1,i0 g3,j0,i1;
```

In[∘]:=
```
θ[{1, i0_, j0_}, {-1, i1_, j1_}] =
  (T3 - 1) g1,j1,i0 g2,i1,i0 g3,j0,i1 - T1^-1 (T3 - 1) g1,j1,j0 g2,i1,i0 g3,j0,i1 -
   (T3 - 1) g1,j1,i0 g2,j1,i0 g3,j0,i1 + T1^-1 (T3 - 1) g1,j1,j0 g2,j1,i0 g3,j0,i1;
```

In[ ]:=
```
θ[{-1, i0_, j0_}, {1, i1_, j1_}] = CF[
   T1^-1 T2^-1 (T3 - 1) (g1,j1,i0 g2,i1,i0 g3,j0,i1 -
      T1 g1,j1,j0 g2,i1,i0 g3,j0,i1 - g1,j1,i0 g2,j1,i0 g3,j0,i1 + T1 g1,j1,j0 g2,j1,i0 g3,j0,i1)];
```

In[ ]:=
```
θ[{-1, i0_, j0_}, {-1, i1_, j1_}] = CF[
   (1 - T3^-1) (-T1^-1 g1,j1,i0 g2,i1,i0 g3,j0,i1 +
      g1,j1,j0 g2,i1,i0 g3,j0,i1 + T1^-1 g1,j1,i0 g2,j1,i0 g3,j0,i1 - g1,j1,j0 g2,j1,i0 g3,j0,i1)];
```

In[ ]:=
```
Γ1[φ_, k_] = -φ / 2 + φ g3,k,k;
```

The Programs

In[ ]:=
```
θ[K_] := Module[{Cs, φ, n, A, s, i, j, k, Δ, G, ν, α, β, gEval, c, z},
   {Cs, φ} = Rot[K]; n = Length[Cs];
   A = IdentityMatrix[2 n + 1];
   Cases[Cs, {s_, i_, j_} :> (A⟦{i, j}, {i + 1, j + 1}⟧ += ({{-T^s, T^s - 1}, {0, -1}}))];
   Δ = T^((-Total[φ]-Total[Cs⟦All,1⟧])/2) Det[A];
   G = Inverse[A]; gEval[ℰ_] := Factor[ℰ /. gν_,α_,β_ :> (G⟦α, β⟧ /. T → Tν)];
   z = gEval[Sum[θ[Cs⟦k1⟧, Cs⟦k2⟧], {k1, 1, n}, {k2, 1, n}]];
   z += gEval[Sum[R1 @@ Cs⟦k⟧, {k, 1, n}]];
   z += gEval[Sum[Γ1[φ⟦k⟧, k], {k, 1, 2 n}]];
   {Δ, (Δ /. T → T1) (Δ /. T → T2) (Δ /. T → T3) z} // Factor];
```

In[ ]:=
```
PolyPlot[0] = Graphics[{}];
PolyPlot[p_] := Module[{crs, m1, m2, maxc, minc, s, hex},
   crs = CoefficientRules[T1^(m1=-Exponent[p,T1,Min]) T2^(m2=-Exponent[p,T2,Min]) p, {T1, T2}];
   maxc = N@Log@Max@Abs[Last /@ crs];
   minc = N@Log@Min@Select[Abs[Last /@ crs], # > 0 &];
   If[minc == maxc, s[_] = 0, s[c_] := s[c] = (maxc - Log@c) / (maxc - minc)];
   hex = Table[{Cos[α], Sin[α]} / Cos[2 π / 12] / 2, {α, 2 π / 12, 2 π, 2 π / 6}];
   Graphics[crs /. ({x1_, x2_} → c_) :> {
       If[c == 0, White, Lighter[If[c > 0, Red, Blue], 0.88 s[Abs@c]]],
       Polygon[(({{1, -1 / 2}, {0, Sqrt[3] / 2}}).{x1 + m1, x2 + m2} + #) & /@ hex] }]];
PolyPlot[{Δ_, θ_}] := PolyPlot[θ]
```

In[∘]:=
```
DunfieldKnots =
  ReadList["../../../People/Dunfield/nmd_random_knots"] /. k_Integer :→ k + 1;
DK[n_] := DunfieldKnots⟦n - 2⟧;
```

In[∘]:=
```
DKString[n_] := StringDrop[ToString[1000 + n], 1]
```

Testing

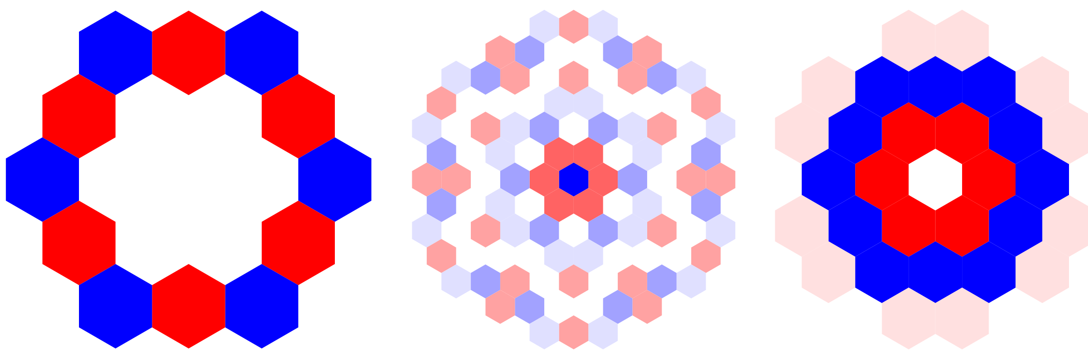
In[∘]:=
```
GraphicsRow[PolyPlot[θ[Knot[#]]⟦2⟧] &
  /@ {"3_1", "K11n34", "K11n42"}]
```

KnotTheory: Loading precomputed data in PD4Knots`.

KnotTheory: Loading precomputed data in DTCode4KnotsTo11`.

KnotTheory: The GaussCode to PD conversion was written by Siddarth Sankaran at the University of Toronto in the summer of 2005.

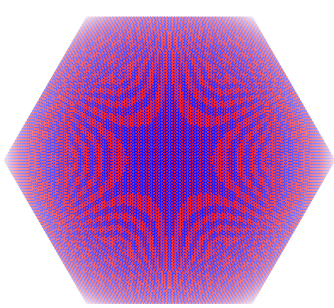
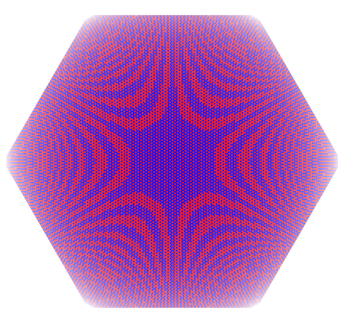
Out[∘]=



Analysis

In[∘]:=
```
{t, θ187} = Get["D187.m"] /. {T1 → T₁, T2 → T₂};
{t, θ210} = Get["D210.m"] /. {T1 → T₁, T2 → T₂};
{Rasterize@PolyPlot[θ187], Rasterize@PolyPlot[θ210]}
```

Out[∘]=

{  ,  }

In[ ]:= **Rasterize@PolyPlot[θ187 - θ210]**

Out[ ]=