Penrose: From Mathematical Notation to Beautiful Diagrams (Supplemental Material)

KATHERINE YE, Carnegie Mellon University
WODE NI, Carnegie Mellon University
MAX KRIEGER, Carnegie Mellon University
DOR MA'AYAN, Technion and Carnegie Mellon University
JENNA WISE, Carnegie Mellon University
JONATHAN ALDRICH, Carnegie Mellon University
JOSHUA SUNSHINE, Carnegie Mellon University
KEENAN CRANE, Carnegie Mellon University

1 LINEAR ALGEBRA

Below, we give the full listing of the linear algebra Domain and Style programs used to create the 2D linear map diagrams in Section 5.4. Note that in the paper, for brevity, we only defined the function add, whereas here we define two functions addV and addS on vectors and scalars, respectively.

The Domain program:

```
-- linear-algebra.dsl
-- Types
type Scalar
type VectorSpace
type Vector
type LinearMap

-- Operators
function neg: Vector v -> Vector
function scale: Scalar c * Vector v -> Vector cv
function addV: Vector * Vector -> Vector
function addS: Scalar s1 * Scalar s2 -> Scalar
function norm: Vector v -> Scalar
function innerProduct: Vector * Vector -> Scalar
function determinant: Vector * Vector -> Scalar
function apply: LinearMap f * Vector -> Vector

-- Predicates
predicate In: Vector * VectorSpace V
predicate From: LinearMap V * VectorSpace domain *
     VectorSpace codomain
predicate Not: Prop p1
predicate Orthogonal: Vector v1 * Vector v2
predicate Unit: Vector v

-- Syntactic sugar
notation "det(v1, v2)" ~ "determinant(v1, v2)"
notation "LinearMap f : U → V" ~ "LinearMap f; From(f, U,
     V)"
notation "v1 + v2" ~ "addV(v1, v2)"
notation "-v1" ~ "neg(v1)"
notation "Vector a ∈ U" ~ "Vector a; In(a, U)"
notation "|y1|" ~ "norm(y1)"
notation "<v1,v2>" ~ "innerProduct(v1, v2)"
notation "s * v1" ~ "scale(s, v1)"
notation "Scalar c := " ~ "Scalar c; c := "
notation "f(v)" ~ "apply(f, v)"
```

The Style program:

```
-- linear-algebra.sty
-- Global constants and sizes
const {
  const.vectorSpaceSize = 300.0
  const.vectorSpaceOffset = 225.0
  const.vectorSpaceGap = 150.0
  const.axisSize = const.vectorSpaceSize * 0.4
  const.distVectorSpaces = 50.0
  const.scaleRatio = 200.0
  const.fontSize = "14pt"
  const.lightenFrac = 0.5
  const.arrowThickness = 2.5
  const.arrowThickness2 = 1.5
  const.arrowheadSize = 0.5
  const.repelWeight = 0.7
  const.textPadding = 15.0
  const.perpLen = 15.0
  -- For unit mark
  const.markerPadding = 15.0
  const.barSize = 5.0
}

-- Global RGB colors in common use
Colors {
```

Authors' addresses: Katherine Ye, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213; Wode Ni, Carnegie Mellon University; Max Krieger, Carnegie Mellon University; Dor Ma'ayan, Technion and Carnegie Mellon University, Haifa, Israel; Jenna Wise, Carnegie Mellon University; Jonathan Aldrich, Carnegie Mellon University; Joshua Sunshine, Carnegie Mellon University; Keenan Crane, Carnegie Mellon University.

```
    Colors.black = rgba(0.0, 0.0, 0.0, 1.0)
    Colors.lightBlue = rgba(0.1, 0.1, 0.9, 0.1)
    Colors.darkBlue = rgba(0.05, 0.05, 0.6, 0.5)
    Colors.lightGray = rgba(200.0,200.0,200.0,1.0)
    Colors.darkGray = rgba(0.4, 0.4, 0.4, 1.0)
    Colors.gray = rgba(0.8, 0.8, 0.8, 1.0)
    Colors.red = rgba(1.0, 0.0, 0.0, 1.0)
    Colors.pink = rgba(1.0, 0.4, 0.7, 1.0)
    Colors.yellow = rgba(1.0, 1.0, 0.0, 1.0)
    Colors.orange = rgba(1.0, 0.6, 0.0, 1.0)
    Colors.lightorange = rgba(1.0, 0.6, 0.0, 0.25)
    Colors.green = rgba(0.0, 0.8, 0.0, 1.0)
    Colors.blue = rgba(0.0, 0.0, 1.0, 1.0)
    Colors.sky = rgba(0.325, 0.718, 0.769, 1.0)
    Colors.lightsky = rgba(0.325, 0.718, 0.769, 0.25)
    Colors.lightblue = rgba(0.0, 0.0, 1.0, 0.25)
    Colors.cyan = rgba(0.0, 1.0, 1.0, 1.0)
    Colors.purple = rgba(0.5, 0.0, 0.5, 1.0)
    Colors.white = rgba(1.0, 1.0, 1.0, 1.0)
    Colors.none = rgba(0.0, 0.0, 0.0, 0.0)
    Colors.bluegreen = rgba(0.44, 0.68, 0.60, 1.0)
    Colors.red2 = hsva(0.0, 100.0, 100.0, 1.0)
    Colors.green2 = hsva(140.0, 23.0, 100.0, 1.0)
}

VectorSpace U {
    U.thickness = 1.5
    U.axisColor = Colors.darkGray
    U.originX = 0.0
    U.originY = 0.0
    U.origin = (U.originX, U.originY)

    U.shape = Square {
        x : U.originX
        y : U.originY
        side : const.vectorSpaceSize
        color : Colors.lightBlue
    }

    U.xAxis = Arrow {
        startX : U.shape.x - const.axisSize
        startY : U.shape.y
        endX : U.shape.x + const.axisSize
        endY : U.shape.y
        thickness : U.thickness
        color : U.axisColor
        arrowheadSize : const.arrowheadSize
    }

    U.yAxis = Arrow {
        startX : U.shape.x
        startY : U.shape.y - const.axisSize
        endX : U.shape.x
        endY : U.shape.y + const.axisSize
        thickness : U.thickness
        color : U.axisColor
        arrowheadSize : const.arrowheadSize
    }

    U.text = Text {
        string : U.label
        x : U.shape.x - const.axisSize
        y : U.shape.y + const.axisSize
        color : U.axisColor
        fontSize : const.fontSize
    }

    U.textX = Text {
        string : "x"
        color : Colors.darkGray
        x : U.xAxis.endX
        y : U.xAxis.endY + const.textPadding
    }

    U.textY = Text {
        string : "y"
        color : Colors.darkGray
        x : U.yAxis.endX + const.textPadding
        y : U.yAxis.endY
    }

    U.perpMark = Square {
        x : U.shape.x + 5.0
        y : U.shape.y + 5.0
        side : 10.0
        color : Colors.white
        strokeColor : Colors.black
        strokeWidth : 1.0
    }

    U.perpLayering1 = U.perpMark below U.xAxis
    U.perpLayering2 = U.perpMark below U.yAxis
    U.perpLayering3 = U.perpMark above U.shape
}

Vector v
with VectorSpace U
where In(v,U) {
  v.text = Text {
    string : v.label
    color : v.shape.color
    fontSize : const.fontSize
  }

  v.color = sampleColor(1.0, "rgb")

  v.shape = Arrow {
    startX : U.shape.x
```

```
    startY : U.shape.y
    endX : ?
    endY : ?

    thickness : const.arrowThickness
    color : v.color
    arrowheadSize : const.arrowheadSize
  }

   v.vector = (v.shape.endX - v.shape.startX, v.shape.endY -
      v.shape.startY)
   v.angle = angle(v.vector)

  v.layeringText1 = v.text above U.xAxis
  v.layeringText2 = v.text above U.yAxis
  v.layeringPerp1 = v.shape above U.perpMark

   v.containFn = ensure contains(U.shape, v.shape)
   v.containLabel = ensure contains(U.shape, v.text)
   v.labelLocation = ensure atDist((v.shape.endX, v.shape.
     endY), v.text, 10.0)
   v.labelAvoidFn = encourage repel(v.shape, v.text, const.
     repelWeight)
}

Vector u
with Vector v; VectorSpace U
where u := neg(v); In(v,U); In(u, U) {
  override u.shape.endX = 2.0 * v.shape.startX - v.shape.
     endX
  override u.shape.endY = 2.0 * v.shape.startY - v.shape.
     endY
}

Vector u
with Vector v; Vector w; VectorSpace U
where u := addV(v,w); In(u, U); In(v, U); In(w, U) {
   override u.shape.endX = v.shape.endX + w.shape.endX - U.
     shape.x
   override u.shape.endY = v.shape.endY + w.shape.endY - U.
     shape.y
   override u.shape.color = blendColor(v.shape.color, w.
     shape.color)

    u.slider_v = Arrow {
        endX : u.shape.endX
        endY : u.shape.endY

        startX : w.shape.endX
        startY : w.shape.endY
        thickness : const.arrowThickness2
        color : scaleColor(v.color, const.lightenFrac)
        style : "dashed"
        arrowheadSize : const.arrowheadSize
    }

    u.slider_w = Arrow {
       endX : u.shape.endX
       endY : u.shape.endY

       startX : v.shape.endX
       startY : v.shape.endY
       thickness : const.arrowThickness2
        color : scaleColor(w.color, const.lightenFrac)
       style : "dashed"
       arrowheadSize : const.arrowheadSize
    }

    u.sw_layering = u.slider_w below u.shape
    u.sv_layering = u.slider_v below u.shape
}

LinearMap f
with VectorSpace U; VectorSpace V
where From(f, U, V) {
   override U.shape.x = U.originX - const.vectorSpaceOffset
   override V.shape.x = U.shape.x + const.vectorSpaceSize +
      const.vectorSpaceGap
   override V.shape.y = U.shape.y

   f.arcRadius = -175.0
   f.left  = [0.65 * U.shape.x + 0.35 * V.shape.x, 0.65 * U.
     shape.y + 0.35 * V.shape.y]
   f.arrowheadLen = 10.0
   f.right = [0.35 * U.shape.x + 0.65 * V.shape.x - f.
     arrowheadLen, 0.35 * U.shape.y + 0.65 * V.shape.y]
   f.center = [(get(f.left, 0) + get(f.right, 0)) / 2.0, f.
     arcRadius]
   f.radius = dist(f.left, f.center)
   f.arcPath = arcPathEuclidean(f.center, f.left, f.right,
     f.radius, "Open")

   f.shape = Curve {
     pathData : pathFromPoints(f.arcPath)
     fill : Colors.none
     color : Colors.black
     strokeWidth : 2.5
     rightArrowhead : True
     arrowheadSize : const.arrowheadSize
   }

   f.labelOffset = 30.0

   f.text = Text {
     string : f.label
     x : (U.shape.x + V.shape.x) / 2.0
     y : U.shape.y + f.labelOffset
     color : f.shape.color
```

```
    fontSize : const.fontSize
  }

  -- define an actual linear map f(x,y) = ax + by
  -- could pick this at random, rather than using a fixed
    map
  f.c0 = (-1.0,0.0)
  f.c1 = (0.0,0.5)

  f.orderFn = ensure lessThan(U.shape.x + 20.0, V.shape.x)
}

-- Scalar is not visualized in any concrete way but instead
     it contains a scalar value
Scalar c {
  c.val = generateRandomReal()
}

Scalar c
with Vector a; Vector b; VectorSpace U
where c := determinant(a, b); In(a, U); In(b, U) {
  c.shape = Curve {
          pathData : makeRegionPath(a.shape, b.shape)
          strokeWidth : 0.0
          fill : setOpacity(Colors.sky, 0.25)
  }

  c.text = Text {
    string : c.label
    -- Normalize for origin
    x : (a.shape.endX + b.shape.endX - a.shape.startX) /
    2.0
    y : (a.shape.endY + b.shape.endY - a.shape.startY) /
    2.0
    color : Colors.sky
    fontSize : const.fontSize
  }

  c.layering = c.text above c.shape
}

Scalar c
with Vector a; Vector b; VectorSpace U
where c := innerProduct(a,b); In(a,U); In(b,U) {
  override c.val = len(a.shape) * calcVectorsAngleCos(a.
    shape.startX, a.shape.startY, a.shape.endX, a.shape.
    endY,  b.shape.startX, b.shape.startY, b.shape.endX, b.
    shape.endY) / 200.0

   c.line1 = Line {
       startX : b.shape.startX
       startY : b.shape.startY
       endX :  b.shape.endX * c.val - b.shape.startX * (c.
    val - 1.0)
       endY : b.shape.endY * c.val - b.shape.startY * (c.
    val - 1.0)
       thickness : 2.5
       style : "dashed"
   }

   c.line2 = Line {
       startX : a.shape.endX
       startY : a.shape.endY
       endX : b.shape.endX * c.val - b.shape.startX * (c.
    val - 1.0)
       endY : b.shape.endY * c.val - b.shape.startY * (c.
    val - 1.0)
       thickness : 2.5
       style : "dashed"
   }
}

Scalar c
with Vector a;  VectorSpace U
where c := norm(a); In(a,U) {
  c.norm = len(a.shape)
  override c.val = c.norm / const.scaleRatio
  c.padding = 20.0

  c.shape = Line {
    startX : a.shape.startX
    startY : a.shape.startY + c.padding
    endX : a.shape.endX
    endY : a.shape.endY + c.padding
    thickness : 1.5
    color : setOpacity(a.shape.color, 0.75)
  }

  c.text = Text {
       string : c.label
       x : c.padding + midpointX(c.shape)
       y : c.padding + midpointY(c.shape)
       color : c.shape.color
       fontSize : const.fontSize
  }
}

Vector u
with Scalar c; VectorSpace U; Vector v
where u := scale(c, v); In(u, U); In(v, U) {
   override u.shape.endX = v.shape.endX * c.val - v.shape.
    startX * (c.val - 1.0)
   override u.shape.endY = v.shape.endY * c.val - v.shape.
    startY * (c.val - 1.0)

   c.layeringScale = u.shape below v.shape
}
```

```
Vector u; Vector v
with VectorSpace U
where Orthogonal(u, v); In(u, U); In(v, U) {
      -- Draw perpendicular mark
      LOCAL.perpMark = Curve {
           pathData : orientedSquare(u.shape, v.shape, U.
     origin, const.perpLen)
           strokeWidth : 2.0
           color : Colors.black
           fill : Colors.white
      }

      -- Make sure vectors are orthogonal
      LOCAL.perpFn = encourage equal(dot(u.vector, v.vector
     ), 0.0)

      LOCAL.layering1 = v.shape above LOCAL.perpMark
      LOCAL.layering2 = u.shape above LOCAL.perpMark
}

-- No label overlaps with any other label, vector, slider
     vector, or axis line.
Vector u1; Vector u2
with VectorSpace U
where In(u1, U); In(u2, U) {
      LOCAL.labelAvoidSegFn = encourage repel(u1.shape, u2.
     text, const.repelWeight)
      LOCAL.labelAvoidTextFn = encourage repel(u1.text, u2.
     text, 1000.0 * const.repelWeight)
}

-- For vectors that are mapped from one vector space to
     another,
-- actually define the target geometry by applying a linear
-- map to the input geometry.  This map is given by the two
      (column)
-- vectors L.c0, L.c1, which together make a matrix.

Vector y
with Vector x; VectorSpace X; VectorSpace Y; LinearMap f
where In(x, X); In(y, Y); From(f, X, Y); y := apply(f, x) {

   -- apply the linear map
   override y.shape.endX = dot( f.c0, x.vector ) + Y.shape.
     x
   override y.shape.endY = dot( f.c1, x.vector ) + Y.shape.
     y

   -- update the underlying vector
   override y.vector = (y.shape.endX - y.shape.startX, y.
     shape.end - y.shape.startY)

   -- also inherit the color
   override y.shape.color = x.shape.color
}

-- v4 = v3, so use its sliders for v3, and hide its label
Vector v4
with Vector u1; Vector u2; Vector u3; Vector v1; Vector v2;
      Vector v3; LinearMap f; VectorSpace U; VectorSpace V
where From(f, U, V); u3 := addV(u1, u2); v1 := apply(f, u1);
      v2 := apply(f, u2); v3 := apply(f, u3); v4 := addV(v1,
      v2) {
      override v4.slider_v.color = u3.slider_v.color
      override v4.slider_w.color = u3.slider_w.color
      override v4.text.string = "\\text{ }"
      delete v4.containFn
      delete v4.containLabel
      delete v4.labelLocation
      delete v4.labelAvoidFn
}

-- v2 = v3, so have its scalar be colored the same, and
     hide its label
Vector v3
with Vector u1; Vector u2; Vector v1; Vector v2; Scalar c;
     LinearMap f; VectorSpace U; VectorSpace V
where From(f, U, V); u2 := scale(c, u1); v1 := apply(f, u1);
      v2 := apply(f, u2); v3 := scale(c, v1) {
      v3.layeringFn1 = v3.shape below v2.shape
      override v3.shape.color = v2.shape.color
      override v3.text.string = "\\text{ }"
      delete v3.containFn
      delete v3.containLabel
      delete v3.labelLocation
      delete v3.labelAvoidFn
}

-- Usually, the unit vector shouldn't need to know about
     orthogonal vectors, but we need to position the unit
     mark so that it doesn't overlap with the "inside" of
     the two vectors
Vector v
with VectorSpace U; Vector w
where In(v, U); Unit(v); Orthogonal(v, w) {

      -- The start and end of the body of the unit marker
     line
      v.offsetVec = unitMark(v.shape, w.shape, "body",
     const.markerPadding, const.barSize)
      v.labelPosition = midpointOffset(v.offsetVec, w.shape,
      const.markerPadding)

      v.unitMarkerLine = Curve {
          pathData : pathFromPoints(v.offsetVec)
          strokeWidth : 2.0
          color : Colors.black
          fill : Colors.none
```

```
    }

    v.unitMarkerEnd1 = Curve {
        pathData : unitMark(v.offsetVec, "start", const.
    markerPadding, const.barSize)
        strokeWidth : 2.0
        color : Colors.black
        fill : Colors.none
    }

    v.unitMarkerEnd2 = Curve {
        pathData : unitMark(v.offsetVec, "end", const.
    markerPadding, const.barSize)
        strokeWidth : 2.0
        color : Colors.black
        fill : Colors.none
    }

    v.unitMarkerText = Text {
        string : "1"
        x : get(v.labelPosition, 0)
        y : get(v.labelPosition, 1)
        color : Colors.black
    }

    v.layeringUnit1 = v.unitMarkerLine above U.xAxis
    v.layeringUnit2 = v.unitMarkerLine above U.yAxis
}
```