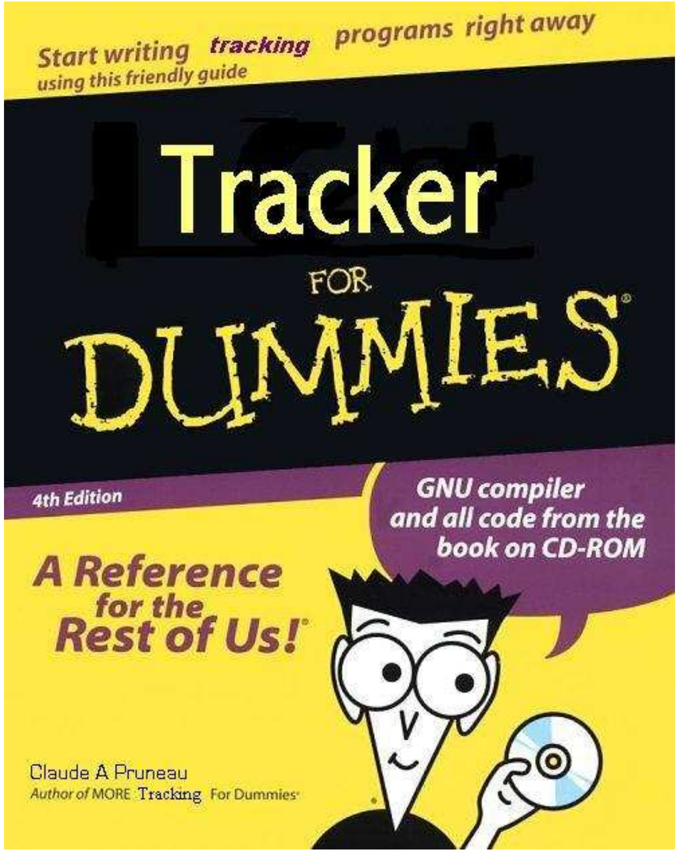
INTRODUCTION TO THE ITTF TRACKER

Claude Pruneau
ITTF Review,
LBNL, June 2003.



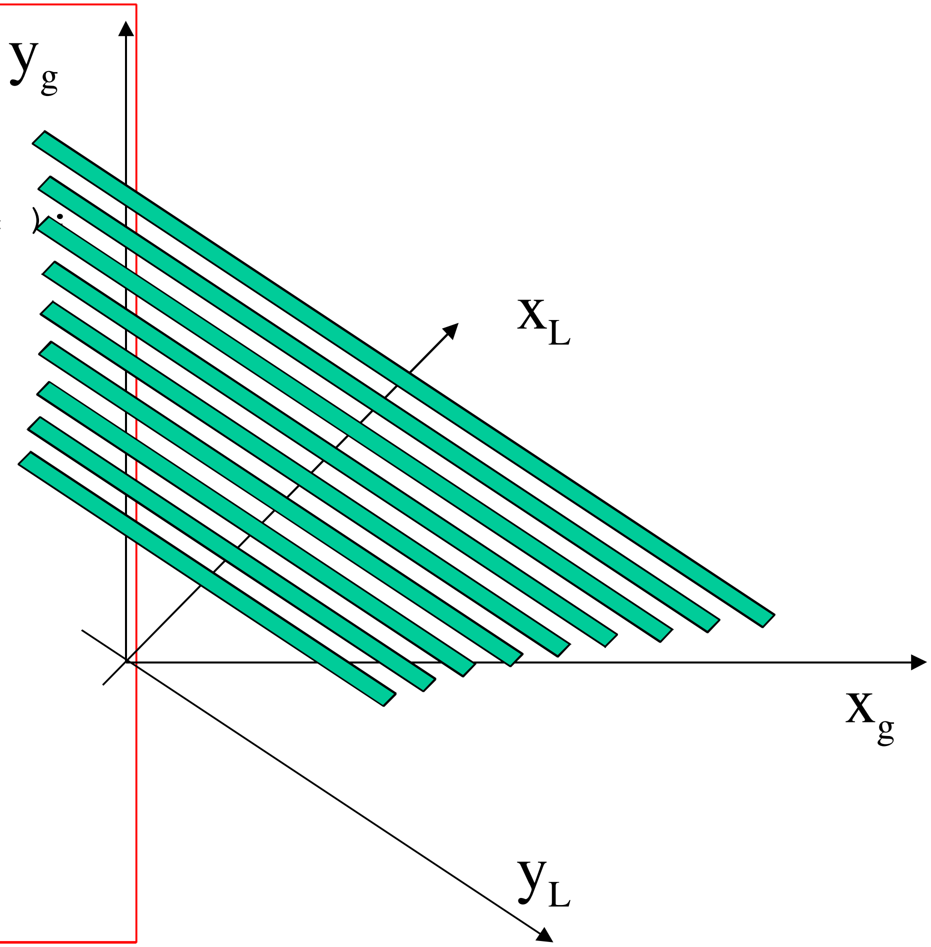
WAYNE STATE UNIVERSITY

Talk Overview

- Hit Model, Representation, and related tools.
- Track Model Representation, and related tools.
- Detector Elements Representation, and related tools.
- Tracker Specifics (parameters)
- How to deploy a new detector system.

Hits : Local & Global Representations

```
class StiHit
{
public:
  enum StiHitProperty
    {kR,kZ,kPseudoRapidity,kPhi};
     StiHit();
    const StiHit& operator=(const StiHit& );
    ///Return the local x value.
     float x() const;
     ///Return the local y value.
     float y() const;
     ///Return the local/global z value.
     float z() const;
     ///Return the local x value.
     float x_g() const;
     ///Return the local y value.
     float y_g() const;
     ///Return the local/global z value.
     float z_g() const;
    …
```



Hits : Practical Matters

```
// class StiHit cont'd
float refangle() const;
///Return the position of the detector plane from which the hit
///arose.
float position() const;
///Return a const pointer to the StiDetector object from which the
///hit arose.
const StiDetector* detector() const;
///Return a const pointer to the StHit object corresponding to this
///StiHit instance
const StMeasuredPoint * stHit() const;
///Return the number of times this hit was assigned to a track
unsigned int timesUsed() const;
```

Hits : Factory – Code Example

```
Factory<StiHit> * hitFactory;

hitFactory = StiToolkit::instance()->getHitFactory();

StiHit * hit;
for (int I=0;I<1000;I++)
{
  hit = hitFactory->getInstance();

// now do something with the hit…

}
```

No memory leaks – Guaranteed!!!

Hits : StiHitContainer

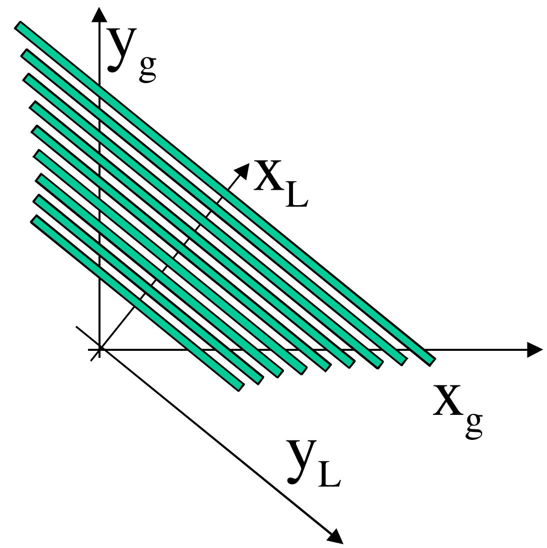
```
_hitContainer->setRefPoint(trackNode);

while (_hitContainer->hasMore())
  {
    stiHit = _hitContainer->getHit();
    if (!stiHit)
      throw logic_error("SKTF::doNextDetector() -F-StiHit*hit==0");
    chi2 = trackNode.evaluateChi2(stiHit);
    if (chi2<_pars->maxChi2ForSelection
          && chi2<trackNode.getChi2())
      {
        trackNode.setHit(stiHit);
        trackNode.setChi2(chi2);
      }
  }
```

Hit Loading : Master Loader

Simplified for illustration

```
template<class Source1, class Source2,class Detector>
void StiMasterHitLoader<Source1, Source2,Detector>::
 loadEvent(Source1 *source1, Source2 * source2, Filter<StiTrack> * trackFilter, Filter<StiHit>   * hitFilter)
{
    // remove all hits currently in the container.
    _hitContainer->clear();
    if (source2)  _mcHitContainer->clear();
    HitLoaderConstIter iter;
    // loop on all subsystems
    for (iter=begin();iter!=end();iter++)
          (*iter)->loadEvent(source1,source2,trackFilter, hitFilter);
    _hitContainer->sortHits();
    _hitContainer->reset();//declare all hits as unused...
    if (source2)
         {
              _mcHitContainer->sortHits();
              _mcHitContainer->reset();
         }
}
```

Track Model



- Reference Frame "Local" to detector
- Representation
  - State vector : $(y,z,\eta,C,\tan\lambda)$\
  - y : track position along virtual pad row
  - z : track position along virtual beam direction
  - $\eta = Cx_o$, where x_o is circle origin.
  - C : circle curvature I.e. 1/R
  - $\tan\lambda$: tangent of helix dip angle

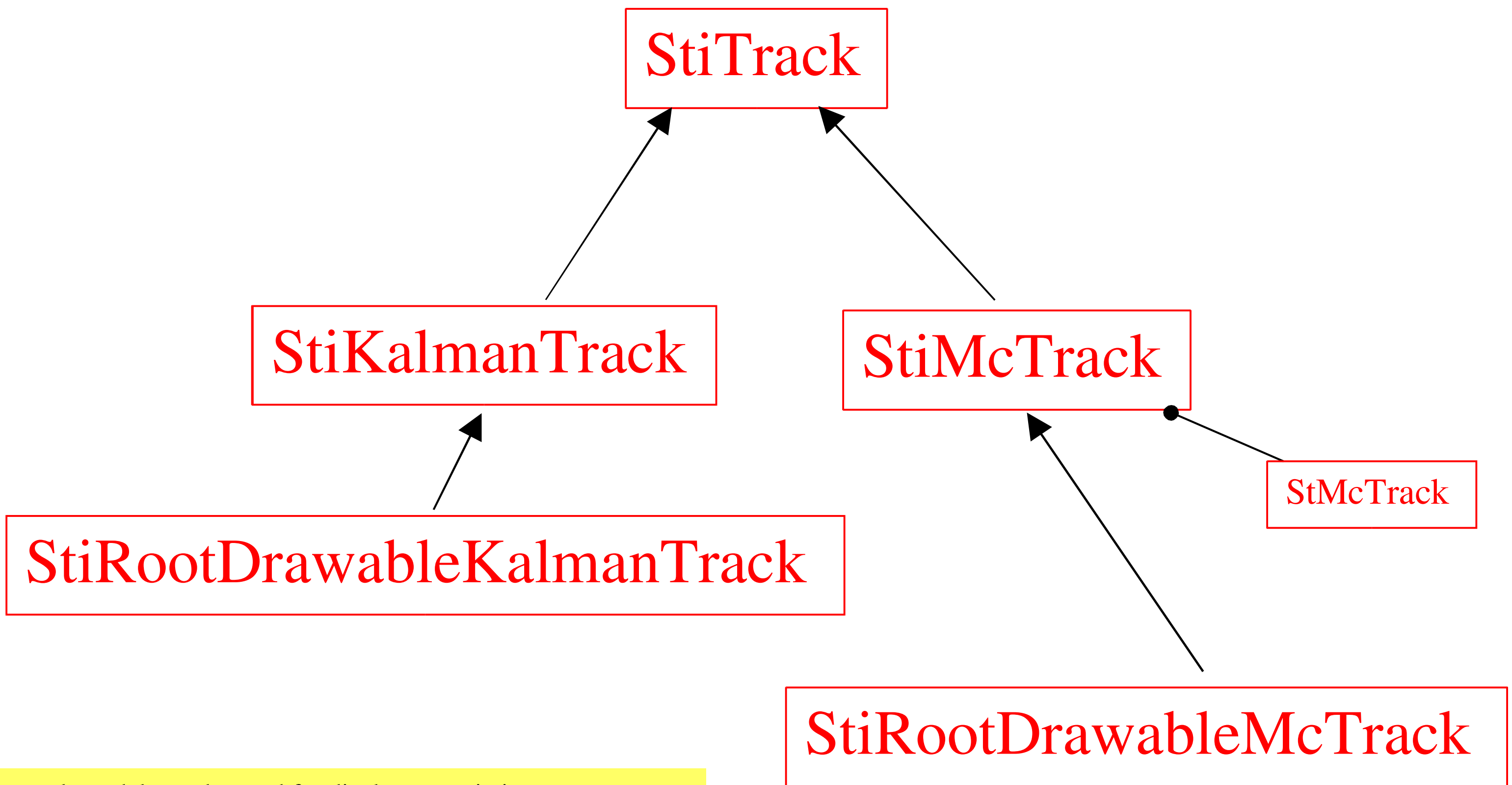
Note : R=infinity implies C=0

Tracks – Abstract Interface

Excerpt – See User guide or Online documentation for details.

```
class StiTrack
{
public:
  virtual void fit(int direction=kOutsideIn);
  virtual bool find(int direction=kOutsideIn);
  virtual void reset()=0;
  virtual void getMomentum(double p[3], double e[6]) const =0;
  virtual StThreeVector<double> getMomentumAtOrigin() const =0;
  virtual StThreeVector<double> getMomentumNear(double x) =0;
  virtual double  getCurvature()      const=0;
  virtual double  getP()              const=0;
  virtual double  getPt()             const=0;
  virtual double  getPseudoRapidity() const=0;
  virtual double  getPhi()            const=0;
  virtual double  getTanL()           const=0;
  virtual int     getPointCount() const=0;
  virtual int     getFitPointCount() const=0;
  virtual int     getGapCount() const=0;
  virtual int     getMaxPointCount() const=0;
```

Reconstructed and MC Tracks



Note: Same track model may be used for display, association, …
More flexible and powerful tools…

Track – Practical Matters

- Reconstructed (Kalman) or Mc Tracks
  - Use same interface
  - Use same containers
  - Use same iterators
- StiKalmanTrack
  - Uses StiKalmanTrackNode for tracking

StiKalmanTrackNode Class

```
class StiKalmanTrackNode : public StiTrackNode
{
public:
void initialize(StiHit*h,double alpha, double eta,
   double curvature,
                      double tanl);
int getCharge() const;
StThreeVector<double> getGlobalMomentum() const;
const StiDetector * getDetector() const;
double getCurvature() const;
double getTanL() const;
double getPt() const;
```

StiKalmanTrackNode Class cont'd

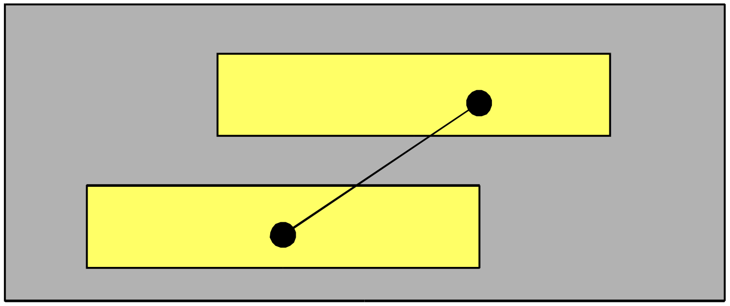
```
int  propagate(StiKalmanTrackNode *p,
               const StiDetector * tDet);
bool propagate(const StiKalmanTrackNode *p,
               StiHit * vertex);
int  propagate(double x,int option);
void propagateError();
void propagateMCS(StiKalmanTrackNode * previousNode,
                const StiDetector * tDet);
void nudge();
double evaluateChi2(const StiHit *hit);
void updateNode();
double getWindowY();
double getWindowZ();
```

Hit Error Calculators

```
class StiHitErrorCalculator
{
 public:
  StiHitErrorCalculator();
  virtual ~StiHitErrorCalculator();
  virtual void calculateError(StiKalmanTrackNode *)const = 0;
};
```

$$\left.\begin{aligned}\Delta y &= \alpha_{y,o} + \alpha_{y,drift}\frac{z-200}{\cos^2\theta} + \alpha_{y,Cross}\tan^2\theta \\ \Delta z &= \alpha_{z,o} + \alpha_{z,drift}\frac{z-200}{\cos^2\lambda} + \alpha_{z,Dip}\tan^2\lambda\end{aligned}\right\}\text{TPC}$$

MCS Propagation



```
double theta2=mcs2(relRadThickness,beta²,p²);
//cout << " theta2:"<<theta2;
double ey  = _p3*_x-_p2;
double ez  = _p4;
double xz  = _p3*ez;
double zz1 = ez*ez+1;
double xy  = _p2+ey;
_c33 += xz*xz*theta2;
_c32 += xz*ez*xy*theta2;
_c43 += xz*zz1*theta2;
_c22 += (2*ey*ez*ez*_p2+1-ey*ey+ez*ez+_p2*_p2*ez*ez)*theta2;
_c42 += ez*zz1*xy*theta2;
_c44 += zz1*zz1*theta2;
```

ELOSS Propagation

$$\left.\begin{aligned} &\Delta E = \\ &\xi = 1 - \frac{E\Delta E}{p^2} \\ &C' = C\xi \\ &\eta' = \eta + xC(\xi - 1) \end{aligned}\right\}$$

Update curvature “C” & $\eta = Cx_o$

```
double eloss = _elossCalculator->calculate(1.,0.5,m, beta2,5.);
double fudge = 1.5;
dE = fudge*sign*dxEloss*eloss;
double cc=_p3;
double correction;
correction =1.- sqrt(e2)*dE/p2;
if (correction>1.1) correction = 1.1;
else if (correction<0.9) correction = 0.9;
_p3 = _p3 *correction;
_p2 = _p2 + _x*(_p3-cc);
```

Bethe-Bloch without delta correction.

Detector Issues

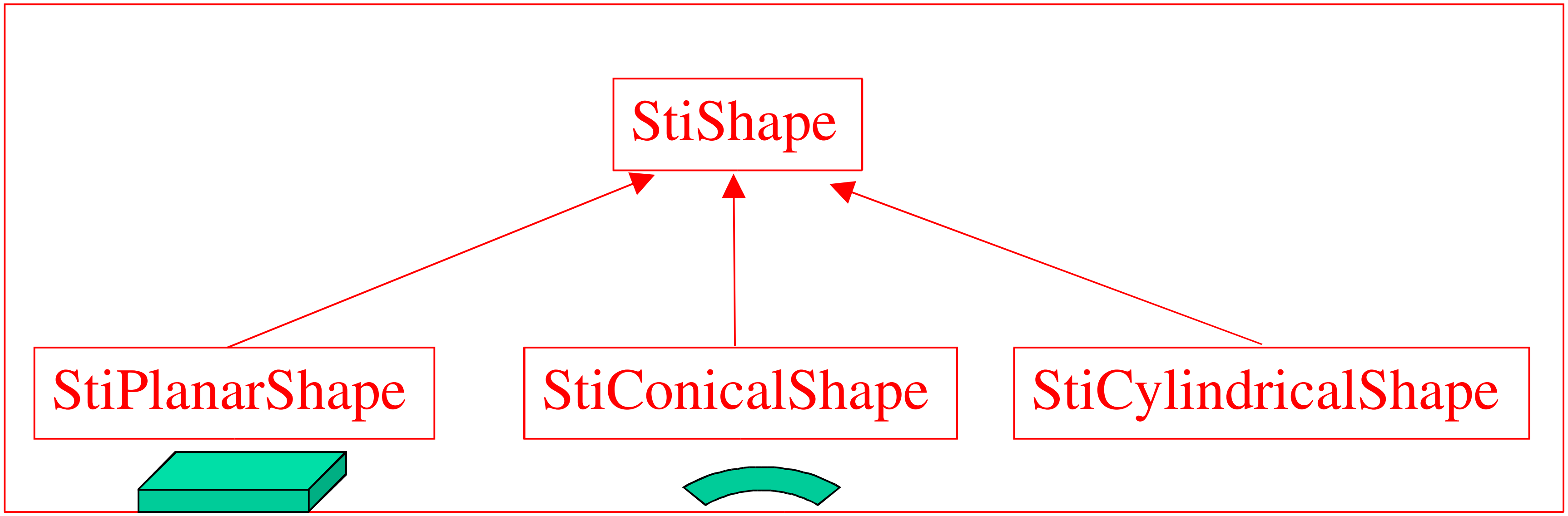
- Representation of detector Modules or elements (e.g. TPC pad row or SVT wafers)
- Geometrical relationship and storage of the detector modules
- Detector traversal (while tracking)
- Detector Builders
- Detector Groups (e.g. TPC, SVT, etc)

Representation of Detector Modules

- Shape and Size
  - Box (Planar), Pipe Section, Cone
- Placement (Position and Orientation).
- Whether it is active (a hit measuring device) or passive.
- Whether it contains a gaseous material and can be considered continuous.
- Whether it is a solid material acting as a discrete scatterer.
- The material composing the bulk of this object.
- The gaseous material, if any, surrounding the object.

Shape and Size

- One Abstract Shape Class
- Three Concrete Shape Classes



Placement (Position and Orientation).

- Key assumptions:
  - N-fold azimuthal (rotational) symmetry.
  - Longitudinal symmetry.
- One concrete class used for positioning and orientation.

```
class StiPlacement
{
 // RTFM for public methods
protected:
    float normalRefAngle; // in [-pi, pi)
    float normalRadius;   // >= 0
    float normalYoffset;
}
```

Materials

- Each Module may have one or two associated materials.
  - Material of the module itself.
  - Material of surrounding gas (e.g. SVT:air).

```
class StiMaterial : public Named
{
public:   // accessor methods not shown…
protected:
    double _z; /// Effective Z
    double _a; /// Effective A
    double _density; /// g/cm^3
    double _radLength; /// radiation length in g/cm^2
    double _ionization; /// Effective ionization (in eV)
    double _zOverA; /// zOverA
    double _x0; /// radiation length in cm.
};
```

StiDetector

- Concrete class used for detector module representation

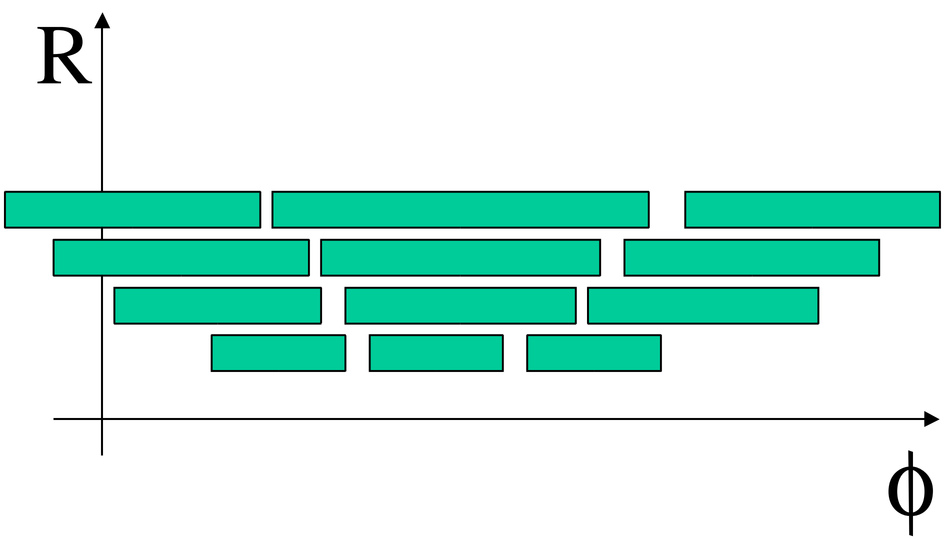
```
class StiDetector : public Named
{
public:
  StiDetector();
    virtual ~StiDetector();
    bool isOn() const {return on;}
    inline bool isActive(double dYlocal, double dZlocal) const;
    inline bool isActive() const;
    StiMaterial* getGas() const { return gas; }
    StiMaterial* getMaterial() const { return material; }
    StiShape* getShape() const { return shape; }
    StiPlacement* getPlacement() const { return placement; }
    void setIsOn(bool val) {on = val;}
    virtual void build();
    const StiHitErrorCalculator * getHitErrorCalculator() const;
protected:
   /// find more by reading the User guide…
};
```

isActive(…) Method

- Active elements produce hits, others do not.
- Active elements may be partly or altogether “broken” or “turned off”.
- Use *delegate* class: StiIsActiveFunctor or one of its derived class to establish the livelihood of an element. Active/Inactive state initialized base on STAR-db at run-time, I.e. when functors are instantiated.
  – State for a given position determined by the initial state and geometry implemented by the functor. Possible to have fractional elements active or inactive (e.g. TPC east/west).

Detector Container

- Geometrical Relationships
  - Above/Below (+r or –r).
  - Next to $+\phi$ or $-\phi$.
  - Next to +z or –z.
- Separate detectors using hierarchy: regions, radius, azimuth.
- Use a Tree, and sorted maps.
- See documentation of StiDetectorContainer class.



Geometry Navigation (while tracking)

- Class StiDetectorContainer
  - Keeps track of "current" position.
  - Has methods "moveIn", "moveOut", "phiPlus", "phiMinus", etc to seek next detector.
- It works, it's fast. Don't want to talk about the details here… Trust us…

Geometry Building

- Use a modular view of STAR.
  - Basic volumes (e.g. beam pipe),
  - Detector groups: SVT, SSD, TPC, etc.
- Geometry deployment at run-time determined by db of each subsystem.
- Instantiation of relevant detector elements (I.e. instances of StiDetector class) and initialization carried on by group builders.
- Registration and invocation of subsystem/group builders taken care of by a master builder.

Geometry Master Builder

- Instance of StiMasterBuilder class
- Registration of subsystem builders performed in StiMaker.
  - Instantiate each required subsystem builder.
  - Add builder to Master builder list.
  - Invoke Master builder "build" method.

Summary of Track Finding Algorithm

Global Tracks Finding

↓

Add/Copy tracks to StEvent as Global Tracks

↓

Find Main Vertex

↓

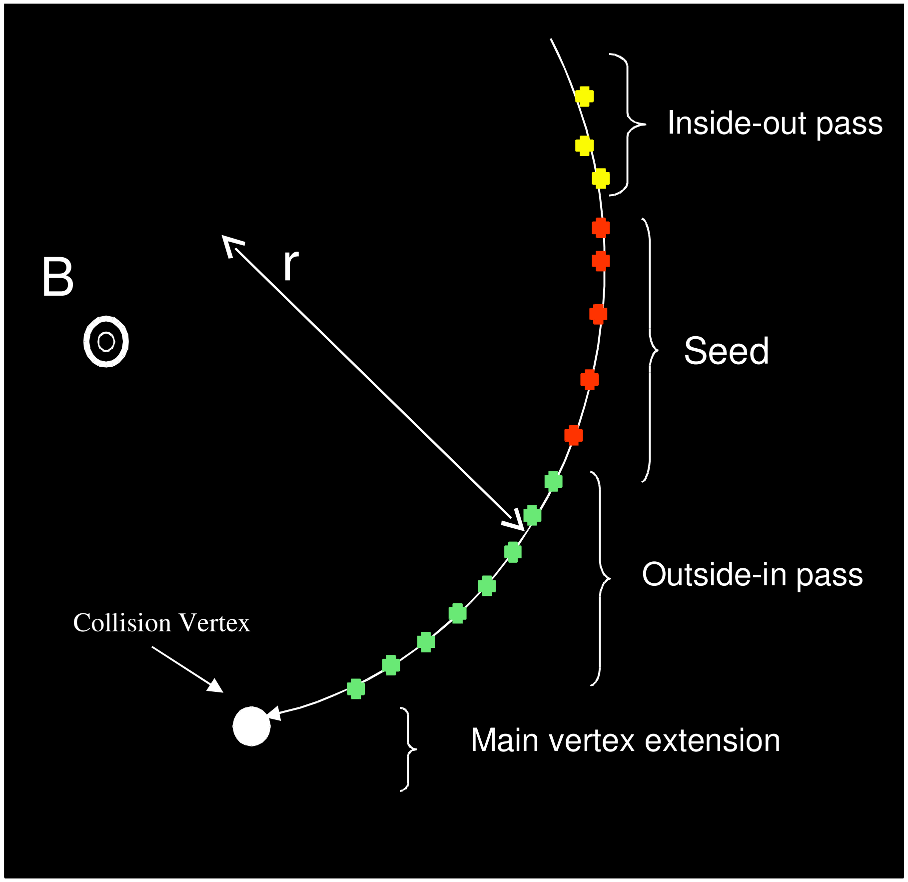
Extend Tracks to Main Vertex, Refit

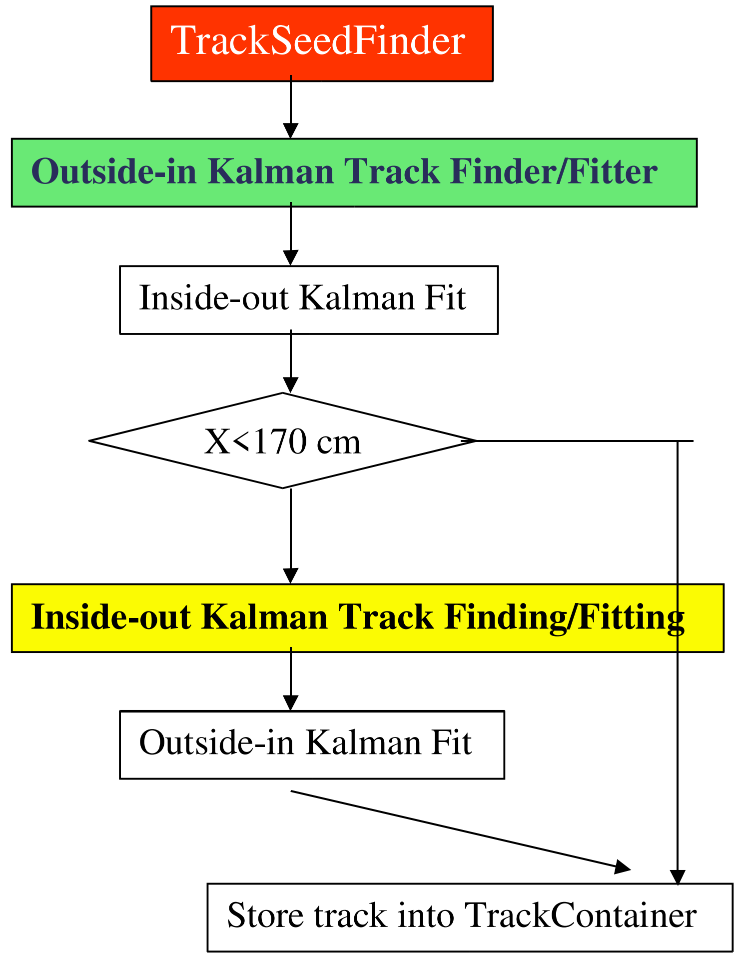
↓

Add/Copy successfully extended tracks to StEvent as Primary Tracks

Output: StEvent
- Global Tracks
- Primary Tracks

Global Track Finding Algorithm





Tracking Parameters

- Minimum search radius (1.5 cm).
- Maximum search radius (7 cm).
- Error Scaling Factors (outer and inner sectors).
- Max Chi2 for inclusion of a point on a track (~8).
- Max Chi2 for inclusion of vertex on a track (~20).
- Max total number of empty “rows” (13).
- Max number of contiguous empty “rows” (8).
- Min number of hits for reset (2).

How to implement/deploy a new detector system?

- Addition of a new software package under StRoot called StiXxx.
- Concrete classes implementations:
  - Class StiXxxDetectorGroup
  - Class StiXxxDetectorBuilder
  - Class StiXxxHitLoader
  - Class StiXxxIsActiveFunctor
  - Class StiXxxHitErrorCalculator
- Modifications to StiMaker/macros/Run.C and StiMaker.cxx files to explicitly include the new detector group.

Concrete classes implementations: Class StiXxxDetectorGroup

- Convenience class used to regrouped all functionalities needed for that detector.

```
class StiXxxDetectorGroup : public StiXxxGroup<StEvent,StMcEvent>
{
  public:
    StiPixelDetectorGroup(bool active);
    ~StiPixelDetectorGroup();
};
```

```
StiXxxDetectorGroup::StiXxxDetectorGroup(bool active)
    : StiDetectorGroup<StEvent,StMcEvent>("Xxx",
                                          active?new StiXxxHitLoader():0,
                                          new StiXxxDetectorBuilder(active),0,0)
{
 /* no operation */
}
```

Concrete classes implementations: Class StiXxxDetectorBuilder

- Has the responsibility to build I.e.instantiate all relevant StiDetector for detector group Xxx.
- Class header:

```
class StiXxxDetectorBuilder : public StiDetectorBuilder
{
public:
   StiXxxDetectorBuilder(bool active);
   virtual ~StiXxxDetectorBuilder();
   virtual void loadDb();
   virtual void buildMaterials();
   virtual void buildShapes();
   virtual void buildDetectors();
   double phiForXxxSector(unsigned int iSector) const;
};
```

```
StiPixelDetectorBuilder::StiPixelDetectorBuilder(bool active)
  : StiDetectorBuilder(“PixelBuilder”,active)
{
   // Parameterized hit error calculator.
   // Given a track (dip, cross, pt, etc) returns average error
   // once you actually want to do tracking, the results depend
   // strongly on the numbers below.
   _hitCalculator = new StiDefaultHitErrorCalculator();
   _ hitCalculator->set(0.00004, 0., 0., 0.00004, 0., 0.);
}
```

```
void StiPixelDetectorBuilder::buildMaterials()
{
 //_gas is the gas that the pixel detector lives in
 _gas= add(new StiMaterial(“Air”,0.49919,1.,0.0012,30420.*0.001205,5.));
 //_fcMaterial is the (average) material that makes up the detector
 // elements.  Here I use ~silicon
 _fcMaterial=add(new StiMaterial(“Si”,14.,28.0855,2.33,21.82,5.) );
}
```

```
void StiPixelDetectorBuilder::buildShapes()
{
 double pixRadius = 5.0; //cm
 StiCylindricalShape *shape = new StiCylindricalShape;
 shape->setName("Pixel/sector");
 shape->setThickness(0.1); //radial thickness in cm
 shape->setHalfDepth( 16./2. ); //half depth along beam direction
 //the angle swept out.  So, for 12 segments, = pi/6
 shape->setOpeningAngle( M_PI/6. );
 shape->setOuterRadius(pixRadius + shape->getThickness()/2.);
 add(shape);
}
```

```cpp
void StiPixelDetectorBuilder::buildDetectors()
{
  char name[50];
  StiShape *shape = findShape("Pixel/sector");
  if (!shape)   throw runtime_error( "StiPixelDetectorBuilder::buildDetectors() - FATAL - shape==0");
  double pixRadius = 5.0;   unsigned int nRows=1;
  StiPlacement *p;
  for (unsigned int row=0; row<nRows; ++row)
   {
    for(unsigned int sector = 0; sector<12; ++sector)        {
                p = new StiPlacement; //see StRoot/Sti/StiPlacement.h
                p->setZcenter(0.);
                p->setLayerRadius(pixRadius);
                p->setRegion(StiPlacement::kMidRapidity);
                p->setNormalRep(phiForTpcSector(sector), pixRadius, 0.);
                StiDetector *detector = _detectorFactory->getInstance();
                sprintf(name, "Pixel/Layer1/Sector_%d", sector);
                detector->setName(name);
                detector->setIsActive(new StiPixelIsActiveFunctor);
                detector->setShape(shape);
                detector->setPlacement(p);
                detector->setGas(_gas);
                detector->setMaterial(_fcMaterial);
                detector->setHitErrorCalculator(_innerCalc);
                add(row, sector, detector); //this hangs it in the right place!
  } } }
```

Concrete classes implementations: Class StiXxxHitLoader

- Load hits of detector Xxx into StiHitContainer.
- Source may be StEvent, StMcEvent, or whatever is appropriate to you.
- Specific Examples Given in User Guide.

```
Class StiTpcHitLoader : public StiHitLoader<StEvent,StMcEvent,StiDetectorBuilder>
{
 public:
  StiTpcHitLoader(StiHitContainer * hitContainer,   StiHitContainer * mcHitContainer,
                                          Factory<StiHit> * hitFactory,     StiDetectorBuilder * detector);
  virtual ~StiTpcHitLoader();
  virtual void loadHits(StEvent* source,Filter<StiTrack>* trackFilter,Filter<StiHit> * hitFilter);
  virtual void loadMcHits(StMcEvent* source,bool useMcAsRec,Filter<StiTrack> * trackFilter,
                                                        Filter<StiHit>*hitFilter);
};
```

Hit Loading Essentials…

```
for (vector<StTpcHit*>::const_iterator iter = hitvec.begin();
                    iter != hitvec.end();
                    iter++)
   {
     StTpcHit*hit=*iter;
     stiHit = _hitFactory->getInstance();
     stiHit->reset();
     stiHit->setGlobal(detector,
                        hit,
                        hit->position().x(),
                        hit->position().y(),
                        hit->position().z(),
                        hit->charge());
     _hitContainer->push_back( stiHit );
    }
```

Concrete classes implementations: Class StiXxxIsActiveFunctor

- Functor has responsibility to determine live state of detector according to hit position.

```
class StiTpcIsActiveFunctor : public StiIsActiveFunctor{
  public:
    /// construct an IsActiveFunctor representing one TPC padrow
    /// spanning both TPC halves.  The sector should be [1-12],
    /// based on the half in the west TPC.  Padrow is in [1-45].
    StiTpcIsActiveFunctor(int iSector, int iPadrow);
    virtual ~StiTpcIsActiveFunctor();
    virtual bool operator()(double dYlocal, double dZlocal);

protected:
    /// returns the RDO board number [1-6] for the tpc padrow [1-45]
    inline static int rdoForPadrow(int iPadrow);
```

StiXxxIsActiveFunctor coding example.

```
bool StiTpcIsActiveFunctor::operator()(double dYlocal, double dZlocal)
{
  if (dZlocal<0.)
    return m_bWestActive && dZlocal>=-200.0;
  else
    return m_bEastActive && dZlocal<= 200.0;
}
```

StiXxxHitErrorCalculator coding example.

```
void StiDefaultHitErrorCalculator::calculateError(StiKalmanTrackNode * node) const
{
 double dz = (fabs(node->getZ())-200.)/100.;
 double cosCA = node->_cosCA;
 double sinCA = node->_sinCA;
 if (cosCA==0.)     cosCA=1.e-10;
 double tanCA = sinCA/cosCA;
 double ecross=coeff[0]+coeff[1]*dz/(cosCA*cosCA) +coeff[2]*tanCA*tanCA;
 double tanDip=node->getTanL();
 double cosDipInv2=1+tanDip*tanDip;
 double edip=coeff[3]+coeff[4]*dz*cosDipInv2+coeff[5]*tanDip*tanDip;
 if (ecross>50) ecross = 50.;
 if (edip>50) edip = 50.;
 double scaling;
 if (node->_x>120)
   scaling = StiKalmanTrackNode::pars->getOuterScaling();
 else
   scaling = StiKalmanTrackNode::pars->getInnerScaling();
 node->eyy = ecross*scaling*scaling; // in cm^2
 node->ezz = edip*scaling*scaling;
}
```

Modifications to StiMaker/macros/Run.C

- Macro carries two arguments for each detector group used in the tracker:
  - "useXXX"
    - specifies whether the given detector group XXX shall be used and instantiated.
  - "activeXXX".
    - specifies whether the detector group Xxx is considered active (useXXX==true) or passive (useXXX=false) while running the tracker.

StiMaker Modification

```
Int_t StiMaker::InitDetectors()
{
  StiDetectorGroup<StEvent,StMcEvent> * group;
  cout<<"StiMaker::InitDetectors() -I- Adding detector group:Star"<<endl;
  _toolkit->add(new StiStarDetectorGroup());
  if (_pars->useTpc)
    {
      cout<<"StiMaker::InitDetectors() -I- Adding detector group:TPC"<<endl;
      _toolkit->add(group = new StiTpcDetectorGroup(_pars->activeTpc));
      group->setGroupId(kTpcId);
    }
 if (_pars->useFtpc)
    {
      cout<<"StiMaker::Init() -I- Adding detector group:FTPC"<<endl;
      _toolkit->add(group = new StiFtpcDetectorGroup(_pars->activeFtpc));
      group->setGroupId(kFtpcWestId);
    }
return kStOk;
}
```