LeptonEfficiencyVsLxyFromMC

Unknown Author

January 17, 2014

Lepton Reconstruction Efficiency vs Lxy from MCWe compute the lepton (e or μ) reconstruction efficiency as a function of the decay length for MC. We use the MC sample: HTo2LongLivedTo4F_MH1000_MFF350_CTau35To3500

We select leptons within the acceptance:

- $p_T > 26$ GeV/c for muons
- $E_T > 40(25)$ GeV and $p_T > 36(21)$ GeV/c for electrons
- $\eta < 2.0$
- $L_{xy} < 50$ cm

and we define the **lepton reconstruction efficiency** as the fraction of times a generated lepton has a matching reconstructed lepton.

We also compute a **dilepton reconstruction efficiency** defined as the fraction of times a generated dilepton decaying to the chosen lepton species is found to have a matching reconstructed dilepton candidate of the same species.

In [35]:
```
import ROOT
from ROOT import TH1F, TEfficiency, TLegend
import rootnotes
import rootprint

ROOT.gROOT.LoadMacro("interface/Loader.C+")

# Bins in the histograms
nBins = 50
```

In [2]:
```
inputFile = ROOT.TFile("/Users/demattia/histograms.root", "READ")
```

The following function checks if the lepton matches the type wanted and if it comes from a long lived particle. The second requirement comes from the fact that we only know how many leptons of the given type were produced from the decay of long lived particles, not the total number of leptons produced in the collision.

In [3]:
```
def isMatchedLepton(lepton, chosenPdgId, signalPdgIds):
    if lepton.isCentralTrack and abs(lepton.genPdgId) == chosenPdgId:
        for signalPdgId in signalPdgIds:
            if lepton.genSignalOriginPdgId == signalPdgId:
                return True
    return False
```

This function fills the numerator and denominator histograms for computing the reconstruction efficiency for leptons within the acceptance as a function of transverse decay length.

In [4]:
```
def computeLeptonEff(event, leptonEffNum, leptonEffDen, chosenPdgId, signa

    # count how many leptons are generated within the acceptance
```

```
    if abs(event.ll1_daughter1_PdgId) == chosenPdgId and event.ll1_decayLe
        if event.ll1_daughter1_Pt > etCut and abs(event.ll1_daughter1_Eta)
            if etCut == ptCut or (not( abs(event.ll1_daughter1_Eta)>1.442
                leptonEffDen.Fill(event.ll1_decayLength2D)
        if event.ll1_daughter2_Pt > etCut and abs(event.ll1_daughter2_Eta)
            if etCut == ptCut or (not( abs(event.ll1_daughter2_Eta)>1.442
                leptonEffDen.Fill(event.ll1_decayLength2D)
    if abs(event.ll2_daughter1_PdgId) == chosenPdgId and event.ll2_decayLe
        if event.ll2_daughter1_Pt > etCut and abs(event.ll2_daughter1_Eta)
            if etCut == ptCut or (not( abs(event.ll2_daughter1_Eta)>1.442
                leptonEffDen.Fill(event.ll2_decayLength2D)
        if event.ll2_daughter2_Pt > etCut and abs(event.ll2_daughter2_Eta)
            if etCut == ptCut or (not( abs(event.ll2_daughter2_Eta)>1.442
                leptonEffDen.Fill(event.ll2_decayLength2D)

    # Compute the efficiency for leptons
    for lepton in event.leptons_:
        if isMatchedLepton(lepton, chosenPdgId, signalPdgIds):
            if lepton.genSignalOriginLxy < LxyCut and lepton.genPt > etCut
            lepton.pt > ptCut and abs(lepton.eta) < etaCut:
                if etCut == ptCut or (lepton.photonEt > etCut):# and \
                                      # abs(lepton.photonEta) < 2.5 ) and
                                      # not( abs(lepton.photonEta)>1.442 a
                                      # not( abs(lepton.genEta)>1.442 and
#                       and ( (abs(leptonL.photonEta) < 1.55 and leptonL
#                       and leptonL.photonHadTowOverEm < maxHoverE_ and 1

                    leptonEffNum.Fill(lepton.genSignalOriginLxy)

    # print "event =", event.candidates.event
    # for candidate in event.candidates.candidates_:
```

This function draws the numerator and denominator histograms

In [5]:
```
def drawCheck(name, leptonEffNum, leptonEffDen):
    leptonEffCheckCanvas = rootnotes.canvas(name, (800, 400))
    leptonEffCheckCanvas.Divide(2,1)
    leptonEffCheckCanvas.cd(1)
    leptonEffNum.Draw()
    leptonEffCheckCanvas.cd(2)
    leptonEffDen.Draw()
    return leptonEffCheckCanvas
```

In [6]:
```
def drawEff(name, leptonEffNum, leptonEffDen):
    leptonEffCanvas = rootnotes.canvas(name, (800, 800))
    leptonEff = TEfficiency(leptonEffNum, leptonEffDen)
    leptonEff.Draw()
    ROOT.gPad.Update()
    leptonEff.GetPaintedGraph().GetYaxis().SetRangeUser(0, 1.1)
    # need to return the TEfficiency or it will be deleted
    return leptonEffCanvas, leptonEff
```

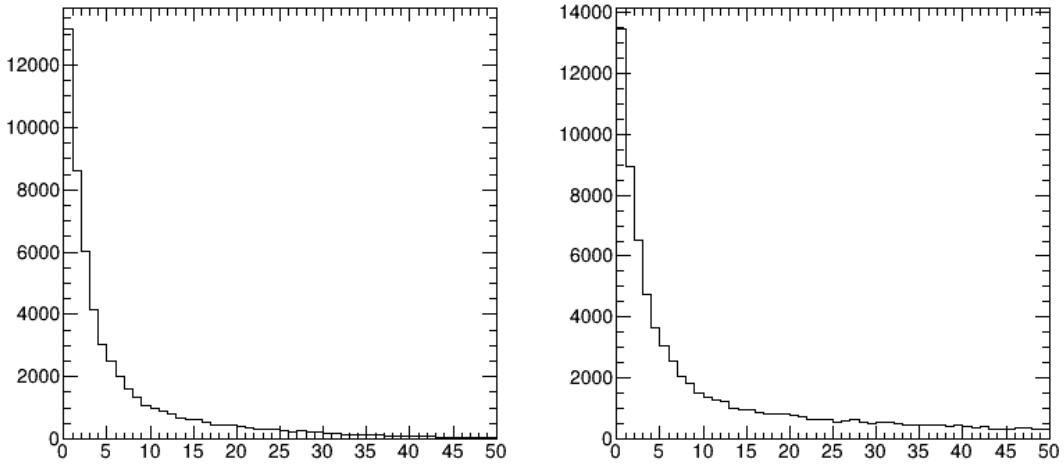
1 Muons

In [7]:
```
directory = inputFile.Get("muTrackAnalysis")
tree = directory.Get("outputTree")

chosenPdgId = 13

signalPdgIds = []
signalPdgIds.append(6001113)
```

```
signalPdgIds.append(6002113)
signalPdgIds.append(6003113)

# Acceptance cuts (for muons)
ptCut = 26.
etCut = ptCut
etaCut = 2.
LxyCut = 50.

leptonEffNum = TH1F("LeptonEffNum", "lepton eff num", nBins, 0, 50)
leptonEffDen = TH1F("LeptonEffDenom", "lepton eff denom", nBins, 0, 50)

# num = 0
for event in tree:
    # num += 1
    computeLeptonEff(event.candidates, leptonEffNum, leptonEffDen, chosenI
    # if num == 10:
    #     break
```

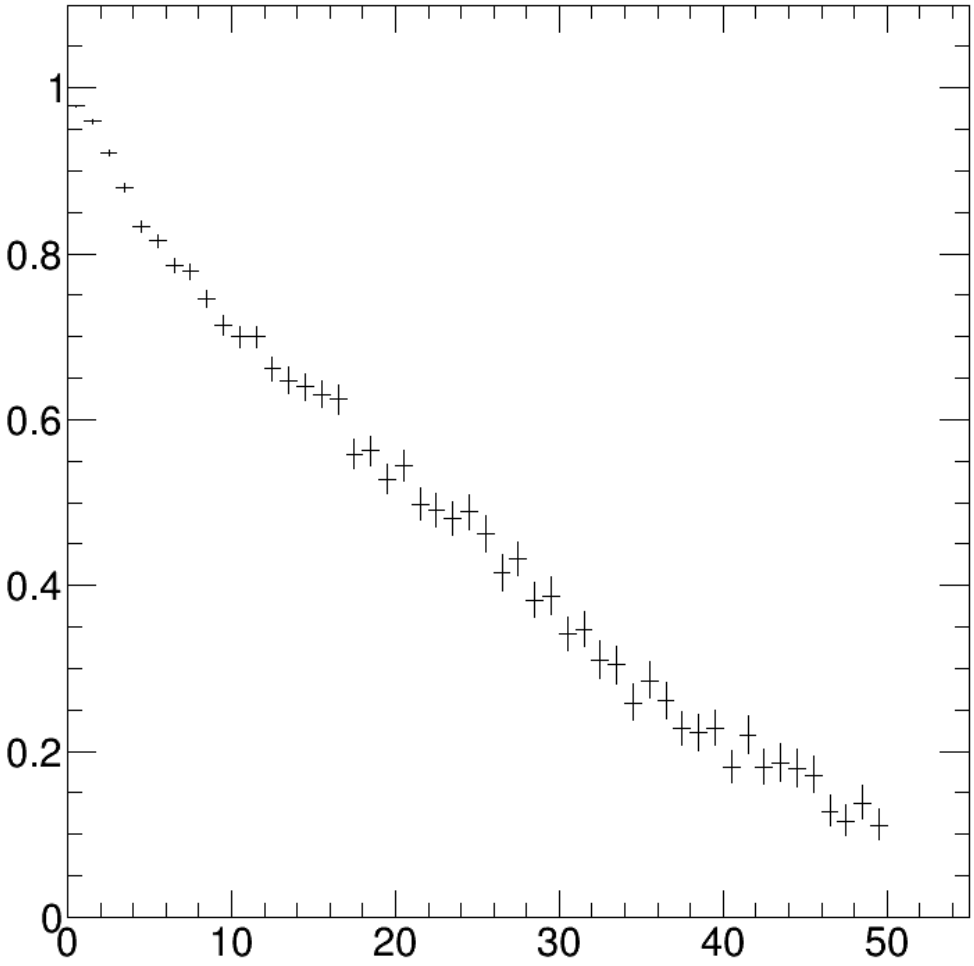
In [8]:
```
drawCheck("LeptonRecoEfficiencyCheckCanvas", leptonEffNum, leptonEffDen)
```

Out [8]:



In [9]:
```
canvasAndEff = drawEff("LeptonRecoEfficiencyCanvas", leptonEffNum, leptonI
canvasAndEff[0]
```

Out [9]:



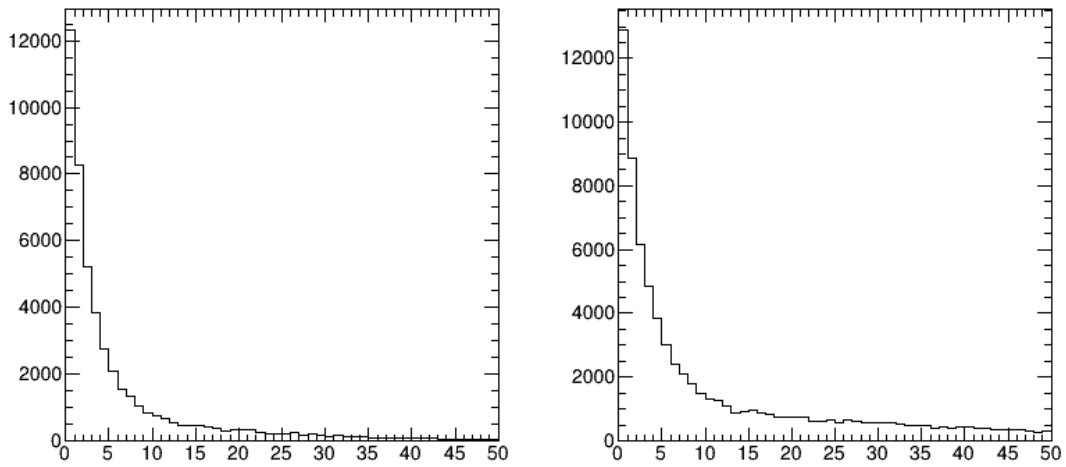
2 Electrons Lower Pt

In [10]:
```
directory = inputFile.Get("eTrackAnalysis")
tree = directory.Get("outputTree")

chosenPdgId = 11
# Acceptance cuts (for electrons)
```

```
ptCutElecLow = 21.
etCutElecLow = 25.

leptonEffNumElecLow = TH1F("LeptonEffNumElecLow", "lepton eff num for lowe
leptonEffDenElecLow = TH1F("LeptonEffDenomElecLow", "lepton eff denom for

# num = 0
for event in tree:
    # num += 1
    computeLeptonEff(event.candidates, leptonEffNumElecLow, leptonEffDenEl
                     chosenPdgId, signalPdgIds, ptCutElecLow, etCutElecLow
    # if num == 100:
    #     break
```

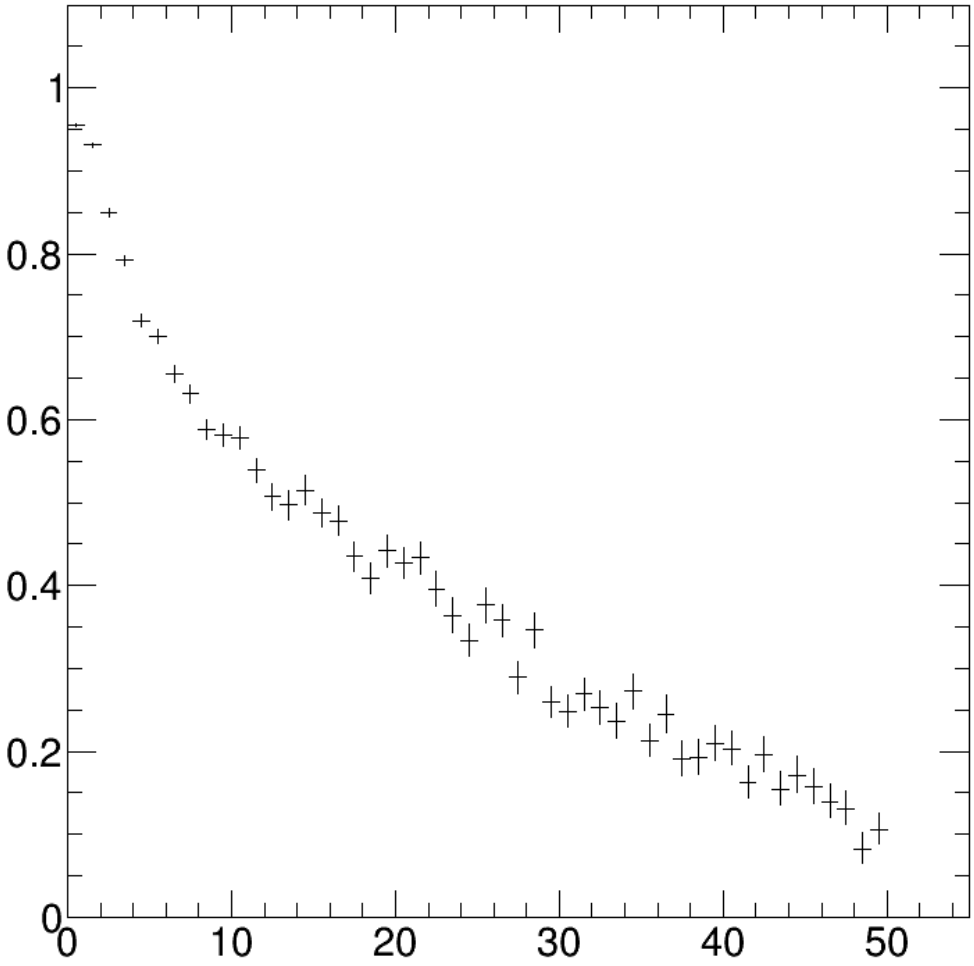
In [11]:
```
canvas = drawCheck("LeptonRecoEfficiencyCheckCanvasElecLow", leptonEffNumE
canvas
```

Out [11]:



In [12]:
```
canvasAndEffElecLow = drawEff("LeptonRecoEfficiencyCanvasElecLow", leptonE
canvasAndEffElecLow[0]
```

Out [12]:

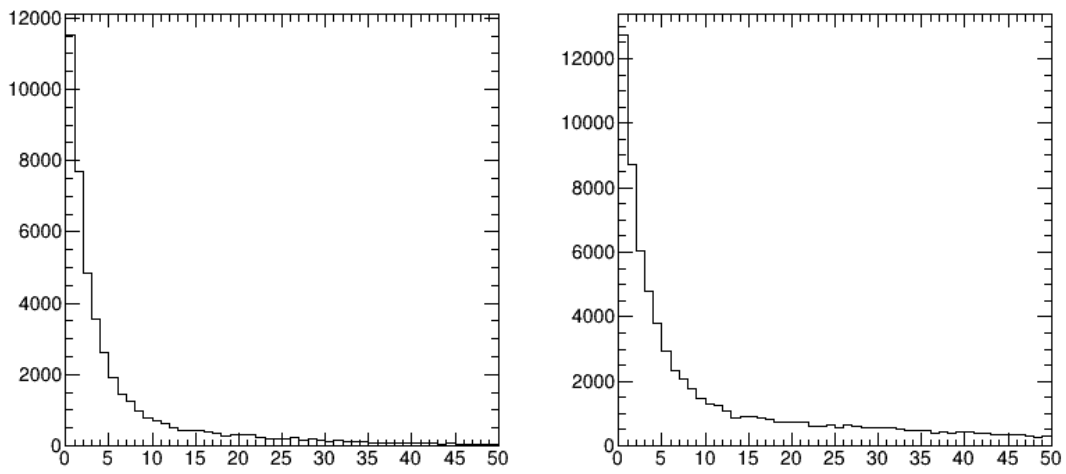


3 Electrons Higher Pt

In [13]:
```
directory = inputFile.Get("eTrackAnalysis")
tree = directory.Get("outputTree")

chosenPdgId = 11
# Acceptance cuts (for electrons)
ptCutElecHigh = 36.
etCutElecHigh = 40.

leptonEffNumElecHigh = TH1F("LeptonEffNumElecHigh", "lepton eff num for hi
leptonEffDenElecHigh = TH1F("LeptonEffDenomElecHigh", "lepton eff denom fo

# num = 0
for event in tree:
    # num += 1
    computeLeptonEff(event.candidates, leptonEffNumElecHigh, leptonEffDenE
                     chosenPdgId, signalPdgIds, ptCutElecHigh, etCutElecHi
    # if num == 100:
    #     break
```

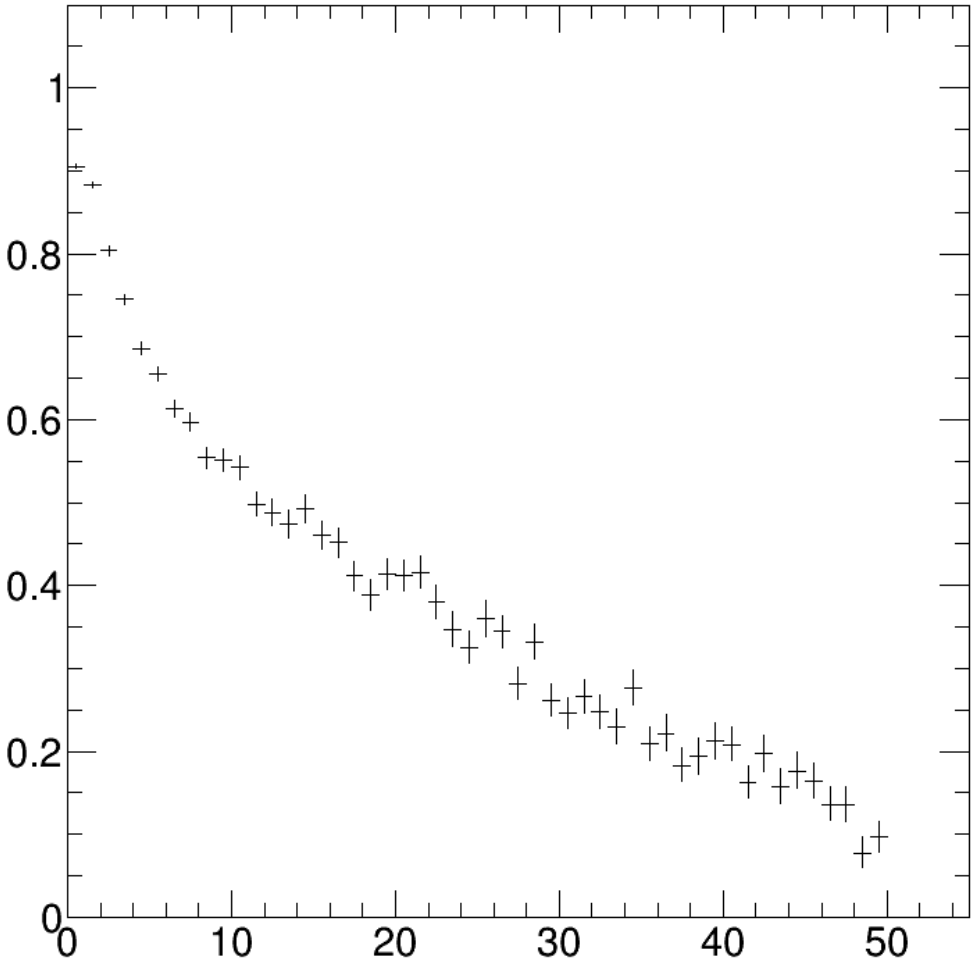
In [14]:
```
canvas = drawCheck("LeptonRecoEfficiencyCheckCanvasElecHigh", leptonEffNur
canvas
```

Out [14]:



In [15]:
```
canvasAndEffElecHigh = drawEff("LeptonRecoEfficiencyCanvasElecHigh", lepto
canvasAndEffElecHigh[0]
```

Out [15]:



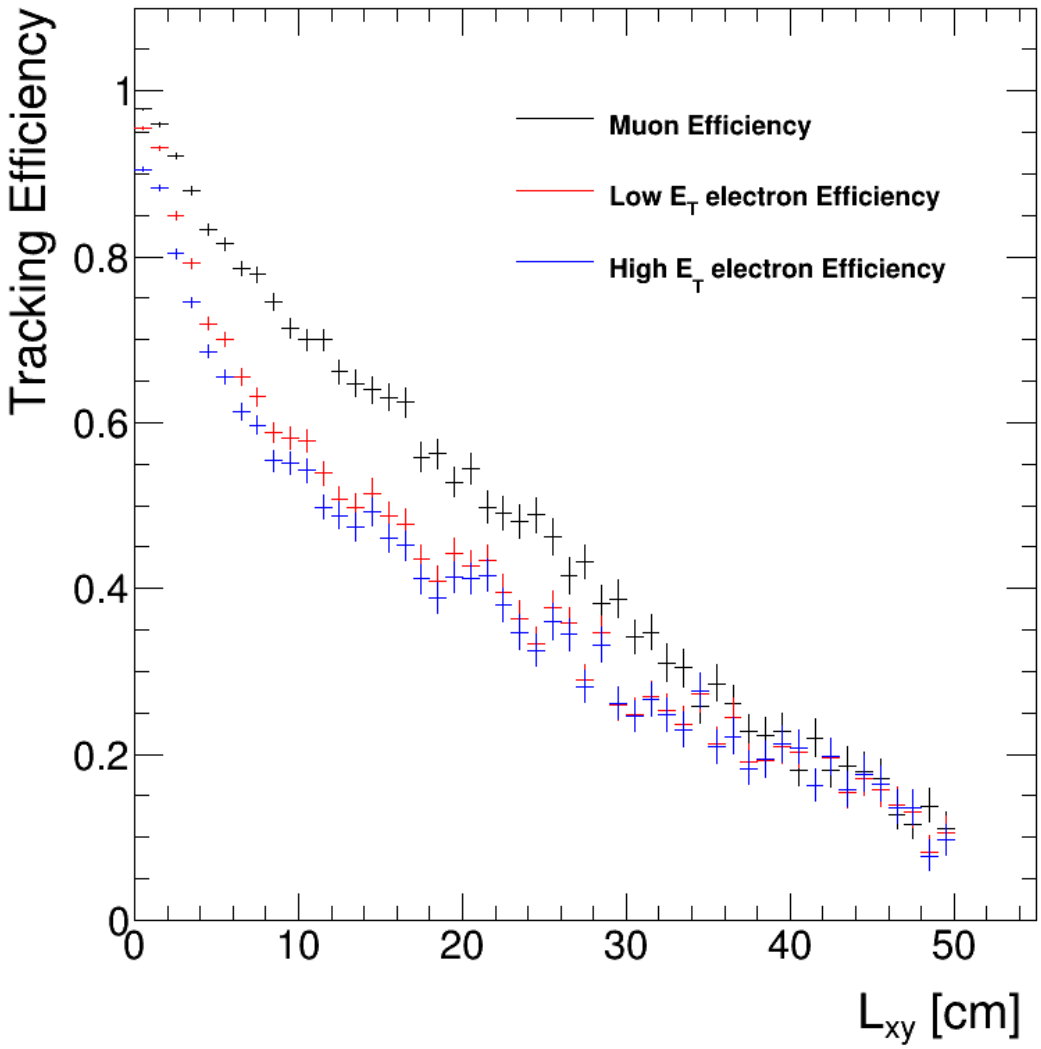
4 Overlay of all efficiencies

In [87]:
```
leptonEffOverlayCanvas = rootnotes.canvas("LeptonEffOverlay", (800, 800))
canvasAndEff[1].Draw()
# canvasAndEff[1].GetPaintedGraph().GetXaxis().SetTitle("L_{xy} [cm]")
# canvasAndEff[1].GetPaintedGraph().GetYaxis().SetTitle("Tracking Efficie
canvasAndEff[1].SetTitle("myTitle; L_{xy} [cm] ; Tracking Efficiency");
ROOT.gPad.Update()
```

```
canvasAndEffElecLow[1].Draw("same")
canvasAndEffElecLow[1].SetLineColor(2)
canvasAndEffElecHigh[1].Draw("same")
canvasAndEffElecHigh[1].SetLineColor(4)
legend = TLegend(0.45,0.7,0.85,0.9)
legend.AddEntry(canvasAndEff[1], "Muon Efficiency", "l")
legend.AddEntry(canvasAndEffElecLow[1], "Low E_{T} electron Efficiency", "
legend.AddEntry(canvasAndEffElecHigh[1], "High E_{T} electron Efficiency",
legend.SetFillColor(0)
legend.SetLineColor(0)
legend.Draw("same")
leptonEffOverlayCanvas
# leptonEffOverlayCanvas.SaveAs("leptonEffOverlay.pdf")
```

```
TCanvas::Constructor:0: RuntimeWarning: Deleting canvas with same
name: LeptonEffOverlay
```

Out [87]:



Print the maximum difference between muon and electron low and high efficiencies

In [91]:
```
muEffMaxDiff = 0

for i in range(1, nBins+1):
    muEff = canvasAndEff[1].GetEfficiency(i)
    diff = max( abs(muEff - canvasAndEffElecLow[1].GetEfficiency(i)), \
                abs(muEff - canvasAndEffElecHigh[1].GetEfficiency(i)) )
    if diff > muEffMaxDiff:
        muEffMaxDiff = diff

print muEffMaxDiff
```

```
0.201467855628
```

The electron systematic can be written as:

$e_{syst} = \mu_{syst} + k \cdot (\epsilon_{\mu_{MC}} - \epsilon_{e_{MC}})$

where, $e_{syst} = \epsilon_{e_{Data}} - \epsilon_{e_{MC}}$, and the same for muons. And ϵ represents the efficiency.

Assuming an uncertainty on the material of 10% ($k = 0.1$), the possible additional variation of the data-MC efficiency difference for electrons is:

In [76]:
```
materialSyst = maxDiff*0.1
materialSyst
```

Out [76]: 0.020146785562804227

Doubling this for a dilepton systematic leads to

In [77]:
```
materialSyst*2
```

Out [77]: 0.040293571125608454

From the tracking systematics in the paper

- 6.1% from cosmics
- 9.8% from track embedding

We get:

In [78]:
```
totalSystWithoutMaterial = ROOT.Math.sqrt(0.061**2 + 0.098**2)
totalSystWithoutMaterial
```

Out [78]: 0.11543396380615197

Adding the material systematic for electrons yields:

In [79]:
```
totalSystWithMaterial = ROOT.Math.sqrt(0.061**2 + 0.098**2 + (materialSyst
totalSystWithMaterial
```

Out [79]: 0.12226435242561287

The variation of the systematic is small, keeping two digits will not change the value from 12%. The relative variation of the systematic is:

In [80]:
```
(totalSystWithMaterial - totalSystWithoutMaterial)/totalSystWithoutMateria
```

Out [80]: 0.059171394572668096

5 Computing the systematic uncertainty for electrons

Definitions:

- $\mu_{syst} = \frac{\epsilon_\mu^{Data} - \epsilon_\mu^{MC}}{\epsilon_\mu^{MC}}$
- $e_{syst} = \frac{\epsilon_e^{Data} - \epsilon_e^{MC}}{\epsilon_e^{MC}}$

Assumption 1: The difference between electrons and muons is determined (to first order) by the material.

Assumption 2: The systematic on the tracking efficiency does not depend on the efficiency. This means that muon and electron systematics will be the same if material effects are perfectly described in the simulation even though the efficiency values can be different.

Note that assumption 2 is likely false. However, if the systematic increases as a function of the efficiency, and because the muon tracking efficiency is generally higher than the one for electrons, this assumption would lead to an overestimate of the systematic for electrons.

We define a as the relative difference of the efficiency for electrons and muons:

$a = \frac{\epsilon_\mu - \epsilon_e}{\epsilon_\mu}$

then

$\epsilon_e = \epsilon_\mu \cdot (1 - a)$

and the relative variation of a between data and MC, which from assumption 1 depends only on the material, is

$k = \frac{a^{MC} - a^{Data}}{a_{MC}}$.

We can compute a^{MC} from our MC efficiencies and k is constrained by the uncertainty on the material (10%).

We want to express the systematic on electrons as a function of the systematic on muons using k and a^{MC}.

From the definition of systematic we can write

$e_{syst} = \frac{\epsilon_e^{Data} - \epsilon_e^{MC}}{\epsilon_e^{MC}} = \frac{(1-a^{Data})\cdot\epsilon_\mu^{Data} - (1-a^{MC})\cdot\epsilon_\mu^{MC}}{(1-a^{MC})\cdot\epsilon_\mu^{MC}}$

By adding and subtracting $a^{MC} \cdot \epsilon_\mu^{Data}$ at the numerator we get

$e_{syst} = \mu_{syst} + \frac{a^{MC} - a^{Data}}{1-a^{MC}} \cdot \frac{\epsilon_\mu^{Data}}{\epsilon_\mu^{MC}}$

Using the fact that $\frac{\epsilon_\mu^{Data}}{\epsilon_\mu^{MC}} = \mu_{syst} + 1$ and that $a^{MC} - a^{Data} = k \cdot a^{MC}$ we get

$e_{syst} = \mu_{syst} + \frac{k \cdot a^{MC}}{1-a^{MC}} \cdot (1 + \mu_{syst})$

We compute this systematic below.Maximum relative difference between muons and electrons efficiency in MC (a^{MC}):

In [122]:
```
effMaxRelDiff = 0

for i in range(1, nBins+1):
    muEff = canvasAndEff[1].GetEfficiency(i)
    diff = max( abs(muEff - canvasAndEffElecLow[1].GetEfficiency(i))/muEf
                abs(muEff - canvasAndEffElecHigh[1].GetEfficiency(i))/muE
    if diff > muEffmaxDiff:
        effMaxRelDiff = diff
```

```
print effMaxRelDiff
```

```
0.132016632017
```

From which we can derive the electron systematic as a function of μ_{syst}, a^{MC} and k:

In [123]:
```
muSyst = ROOT.Math.sqrt(0.061**2 + 0.098**2)
k = 0.1
materialTerm = k*effMaxRelDiff/(1-effMaxRelDiff)
eSyst = muSyst + materialTerm*(1+muSyst)
eSyst
```

Out [123]:
```
0.13239924684847307
```

This is ~1% bigger than μ_{syst}.

In []:
```
```