# - Relatório Técnico -

Este é o quarto relatório do projeto **TH-APP** e representa as atividades executadas durante o terceiro mês.

As atividades executadas foram:

* Aplicação das estratégias de segmentação do grafo para geração de grandes grafos através da operação de Geração de Fluxos Locais ,
* Utilização das estratégias de segmentação do grafo para visualização de grandes grafos armazenados em bancos de dados.

## Introdução

Este relatório apresenta os resultados obtidos utilizando o que foi definido no relatório anterior. São utilizados as estratégias de *cache* e recuperação/armazenamento dos dados de um grafo para fazer a sua segmentação, permitindo assim a manipulação de grandes grafos. A segmentação do grafo é representado pelos pacotes de dados que estão presentes no *cache*. Como esses pacotes são preenchidos é definido pela estratégia de busca adotada.

## Estratégia de *Cache*

Podemeos entender o *cache* como um *container* que possui um histórico do que foi acessado. A eficiência desse *cache*  esta na definição do seu tamanho e na lógica do que deve ser mantido em memória e o que deve ser descartado.

Dentro da lógica desenvolvida para a manipulação de grafos na TerraLib, duas variáveis são utilizadas para definir o *cache*.

* Número de pacotes no *cache* do grafo
  + Valor *default*: 5 - Variável: TE\_GRAPH\_DEFAULT\_MAX\_VEC\_CACHE\_SIZE
* Número de elementos em um pacote
  + Valor default: 100.000 - Variável: TE\_GRAPH\_DEFAULT\_MAX\_CACHE\_SIZE

## Lógica do Cache

A lógica do *cache* é definida para três operações:

* Adição: Ao adicionar um novo elemento ao grafo é necessário verificar se tem algum pacote no *cache* para adicionar esse elemento, além disso esse pacote não pode estar com sua capacidade atingida.
* Busca: Ao buscar por um elemento do grafo é necessário verificar nos pacotes em memória se esse elemento já esta presente no *cache*, caso contrário é necessário buscar esse elemento na fonte de dados, figura abaixo.
* Remoção: A escolha do pacote a ser removido dependerá da estratégia adotada.

### acesso_dados_color

### Cache - Adição

Para que um novo elemento seja adicionado ao grafo, o cache precisa retornar uma pacote (GraphData) válido para que o elemento seja adicionado, abaixo é apresentado a lógica utilizada:

te::graph::GraphData\* te::graph::GraphCache::getGraphData()

{

//cache is empty, return a new graph data

if(m\_graphDataMap.empty())

{

return createGraphData();

}

//verify if has a incomplete graph data in cache and return

std::map<int, GraphData\*>::iterator itMap = m\_graphDataMap.begin();

int gdId = -1;

size\_t maxSize = m\_metadata->m\_maxCacheSize - 1;

while(itMap != m\_graphDataMap.end())

{

te::graph::GraphData\* d = itMap->second;

if (d->getVertexMap().size() < maxSize && d->getEdgeMap().size() < maxSize)

{

gdId = d->getId();

maxSize = d->getVertexMap().size();

}

++itMap;

}

if(gdId != -1)

{

m\_policy->update(gdId);

return m\_graphDataMap[gdId];

}

//if cache is not full create a new graph data... add to cache and return

if(m\_graphDataMap.size() < m\_metadata->m\_maxVecCacheSize)

{

return createGraphData();

}

//remove the a graph data following the cache policy and create a new graph data

if(m\_graphDataMap.size() == m\_metadata->m\_maxVecCacheSize)

{

int idxToRemove = 0;

m\_policy->toRemove(idxToRemove);

te::graph::GraphData\* d = m\_graphDataMap[idxToRemove];

if (d)

{

saveGraphData(d);

removeGraphData(d->getId());

}

return createGraphData();

}

return 0;

}

### Cache - Busca

A busca por um elemento é realizada primeiramente pesquisando-se os pacotes em memória, caso não seja encontrado, é utilizado a estratégia de busca para trazer um novo conjunto de elementos para o *cache*.

A mesma lógica utilizada para buscar um vértice de um grafo é aplicada para recuperar uma aresta. O pacote presente no *cache* é composto por um conjunto de vértices e um conjunto de arestas.

te::graph::GraphData\* te::graph::GraphCache::getGraphDataByVertexId(int id)

{

//check local cache

std::map<int, GraphData\*>::iterator itMap = m\_graphDataMap.begin();

while(itMap != m\_graphDataMap.end())

{

te::graph::GraphData\* d = itMap->second;

if (d)

{

te::graph::GraphData::VertexMap::iterator it = d->getVertexMap().find(id);

if (it != d->getVertexMap().end())

{

m\_policy->accessed(d->getId());

return d;

}

}

++itMap;

}

//if not found

if(m\_dataManager != 0)

{

m\_dataManager->loadGraphDataByVertexId(id, this);

return checkCacheByVertexId(id);

}

return 0;

}

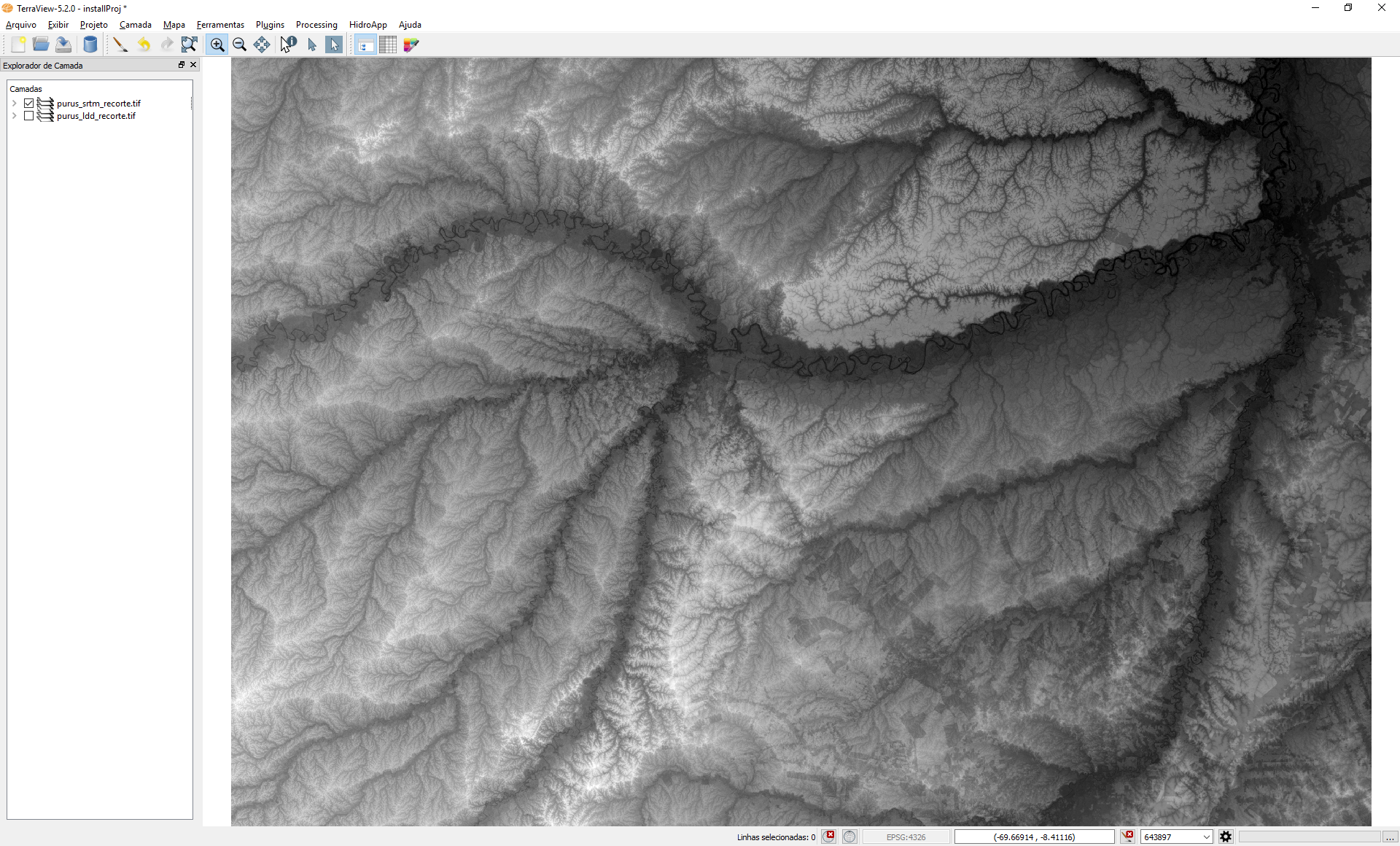
## Resultados da Geração de Fluxos Locais e desenho do grafo

Como resultado da utilização do modelo de *cache* e carga dos dados implementado, é feito o processamento de extração do grafo a partir de uma imagem representando os fluxos locais.

A imagem utilizada é um recorte da Bacia de Purus, imagem abaixo, que tem as seguintes características:

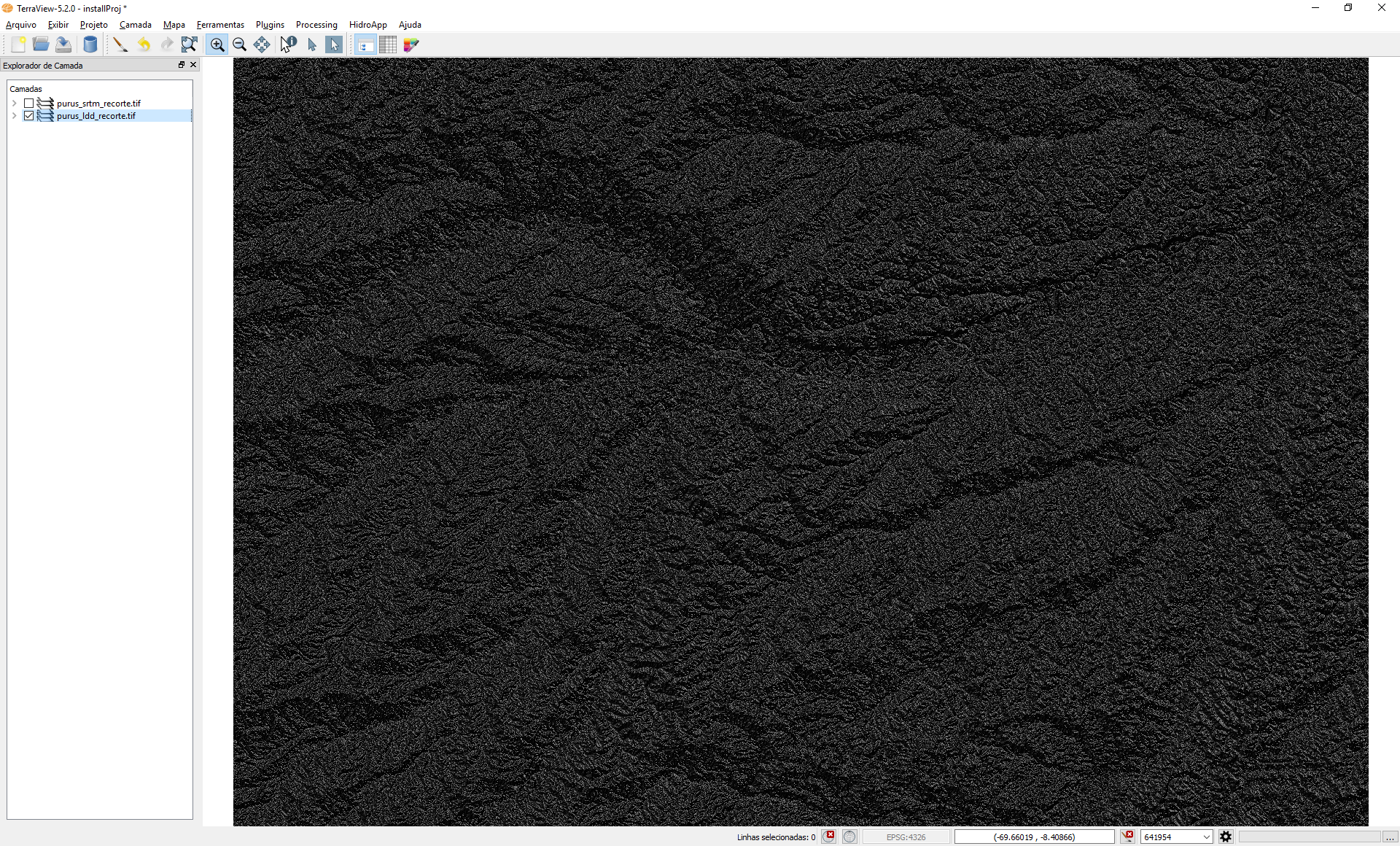
* Linhas: 1.993
* Colunas: 2.955

SRTM



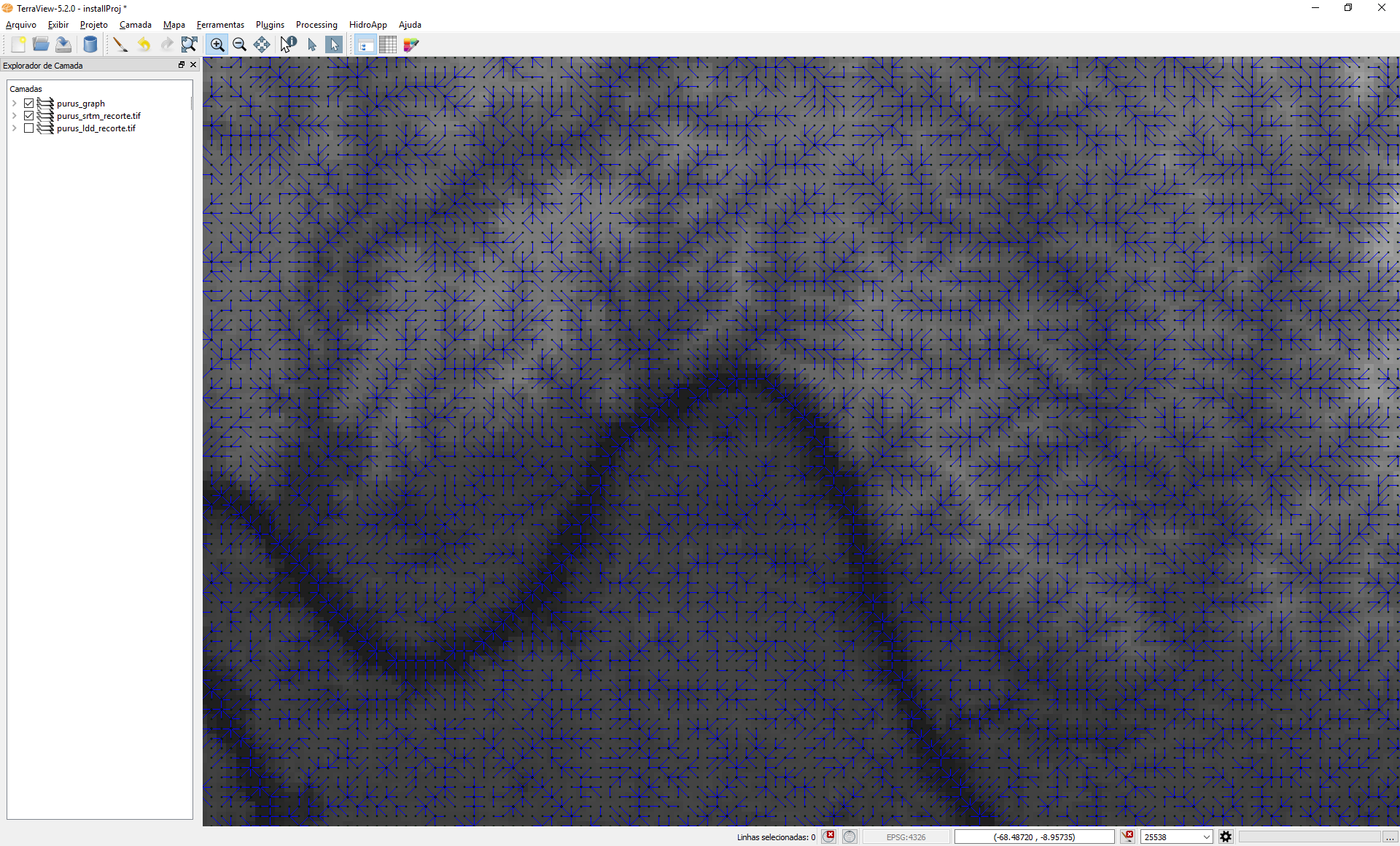
Os fluxos locais (LDD) da bacia de Purus é apresentado abaixo, também possui o mesmo número de linhas e colunas da imagem SRTM.

LDD

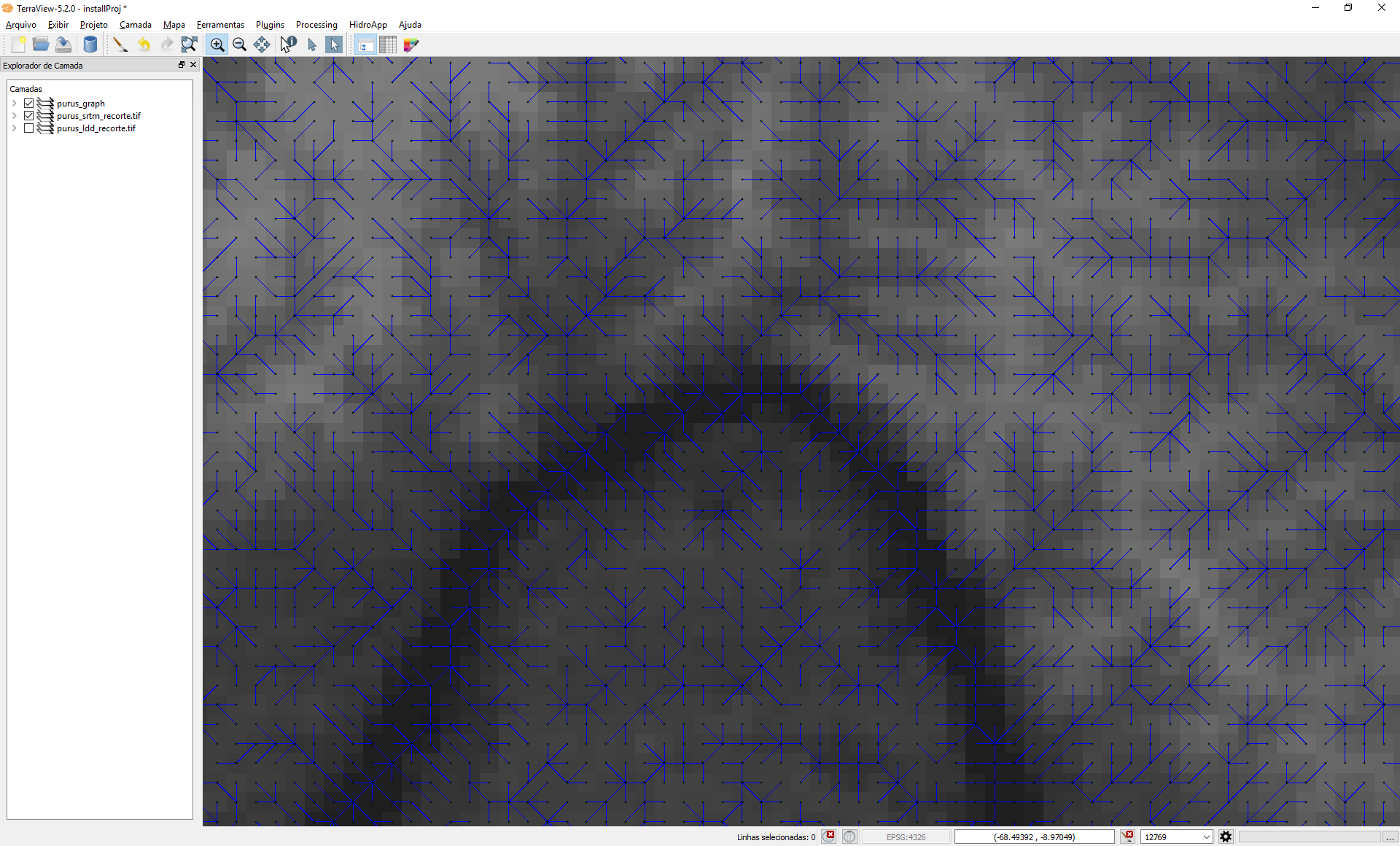


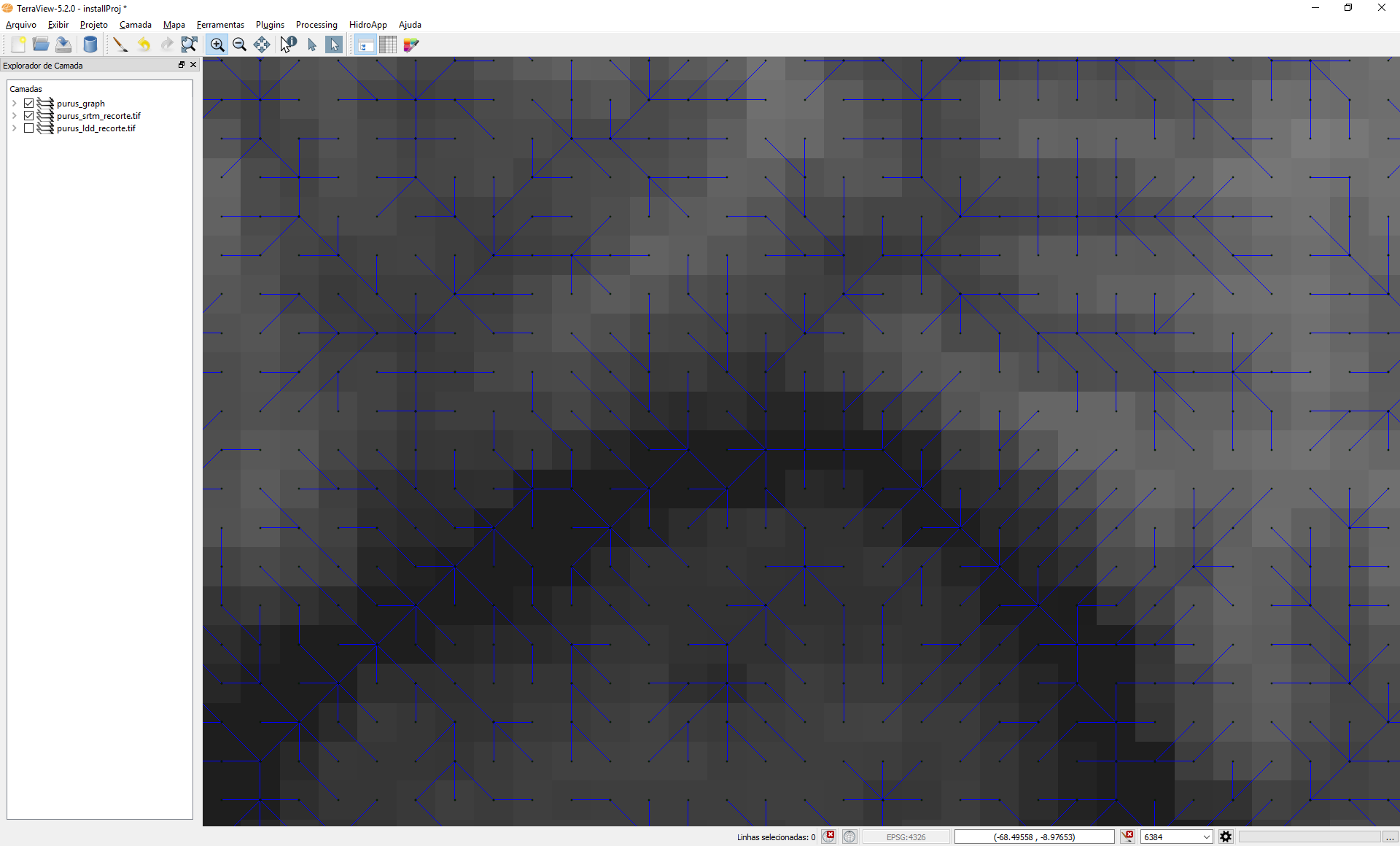
O grafo resultante da extração dos fluxos locais é um grafo com as seguintes características:

* Vértices: 5.822.893
* Arestas: 5.818.234



Abaixo seguem outras visualizações do grafo em escalas diferentes.





Esse é um exemplo de grafo que não seria possível sua manipulação sem a estratégias de *cache*, devido ao grande número de elementos. Para este teste foram utilizados os valores padrões de tamanho de *cache*.